

Rochester Institute of Technology

RIT Scholar Works

Theses

12-2017

Design of a Single Precision Floating Point Divider and Multiplier with Pipelined Architecture

Mayuresh Vijay Keni
mvk3029@rit.edu

Follow this and additional works at: <https://scholarworks.rit.edu/theses>

Recommended Citation

Keni, Mayuresh Vijay, "Design of a Single Precision Floating Point Divider and Multiplier with Pipelined Architecture" (2017). Thesis. Rochester Institute of Technology. Accessed from

This Master's Project is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

DESIGN OF A SINGLE PRECISION FLOATING POINT DIVIDER AND MULTIPLIER
WITH PIPELINED ARCHITECTURE

by
Mayuresh Vijay Keni

GRADUATE PAPER

Submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE
in Electrical Engineering

Approved by:

Mr. Mark A. Indovina, Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
DECEMBER 2017

To my Mom, Sister and Friends, for all of their endless love, support, and encouragement
throughout my career at Rochester Institute of Technology.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This paper is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Mayuresh Vijay Keni

December 2017

Acknowledgements

I would like to thank my adviser, professor, and mentor, Mark A. Indovina, for all of his guidance and feedback throughout the entirety of this project. He is the reason for my love of digital hardware design and drove me to pursue it as a career path. He has been a tremendous help and a true friend during my graduate career at RIT.

Another professor I would like to thank is Dr. Dorin Patru. He cleared my computer architecture knowledge and always provided helpful feedback for my random questions.

I would also like to thank Akshay, Mohit, Surya and Saurabhi for guiding me and provide great company throughout many sleepless nights spent on this project.

Finally, I would like to thank my best friends, Vishal, Aniket, Prashant and Shishir, without their advice, love and guidance my career at RIT wouldn't have been possible.

Abstract

High speed computation is the need of today's generation of Processors. To accomplish this major task, many functions are implemented inside the hardware of the processor rather than having software computing the same task. Majority of the operations which the processor executes are Arithmetic operations which are widely used in many applications that require heavy mathematical operations such as scientific calculations, image and signal processing. Especially in the field of signal processing, multiplication division operation is widely used in many applications. The major issue with these operations in hardware is that many iteration's are required which results in slow operation while fast algorithms require complex computations within each cycle. The result of a Division operation results in a either in Quotient and Remainder or a Floating point number which is the major reason to make it more complex than Multiplication operation. The work described in this paper includes design and verification of a floating point divider and multiplier. The inputs of both the Multiplier and Divider and also the output are designed using the single precision IEEE Standard for floating point numbers.

Contents

Declaration	ii
Acknowledgements	iii
Contents	v
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Organization	2
2 Background Research	3
2.1 Shift and Subtract Divider	4
2.2 Newton-Raphson Algorithm	4
2.3 Liddicoat and Flynn Divider	7
3 Design and Algorithm	11
3.1 P. Hung's Divider using Taylor Series	11
3.1.1 P. Hung's Algorithm	12
3.1.2 Error Analysis of P. Hung's Algorithm	14
3.2 Single Precision Floating Point Divider Algorithm	15
3.2.1 IEEE-754 Standard for Floating Point Numbers	15
3.2.1.1 Floating Point Format	16
3.2.1.2 Not A Number (NaN)	18
3.2.2 Algorithm and Architecture	18
3.2.3 Error Analysis	23
4 Wallace Tree Multiplier	25
4.1 Algorithm	28

5	Results	31
5.1	Floating Point Divider	31
5.2	Floating Point Multiplier	31
6	Conclusions	35
6.1	Conclusions	35
6.2	Future Work	35
	References	37
I	Source Code	I-1
I.1	Floating Point Divider	I-1
I.2	Floating Point Multiplier	I-6
I.3	Wallace Tree Multiplier	I-32
I.4	Floating Point Subtractor	I-91
I.5	Subtractor Module	I-109

List of Figures

2.1	Basic Shift Divider[1]	5
2.2	Newton - Raphson's Divider [2]	6
2.3	Liddicoat and Flynn Divider [2]	8
3.1	P. Hung's Divider Algorithm using Taylor Series [3]	13
3.2	Single Precision Floating Point Format[4]	16
3.3	Double Precision Floating Point Representation[4]	17
3.4	Architecture of the Divider	22
4.1	Basic Multiplier [5]	26
4.2	Booth's Algorithm	27
4.3	8 Bit Multiplier Using CLA Adder [5]	28
4.4	8 Bit Wallace Tree Multiplier	29
5.1	32nm Divider Result	32
5.2	65nm Divider Result	33
5.3	180nm Divider Result	33

List of Tables

5.1	Area, Power, Timing, and DFT Coverage of Divider	32
5.2	Area, Power, Timing, and DFT Coverage of Multiplier	34

Chapter 1

Introduction

The most commonly used floating point formats are Single precision and Double precision floating point format. Both of these formats are defined by the IEEE 754-2008 standard where single precision format consists of a 32 bit binary number and the double precision format consists of a 64 bit binary number. Both of these numbers are divided into different sections which together represents a floating point value. In this paper, a 32 bit Single Precision Floating Point Divider and Multiplier is designed using pipelined architecture. A 32 bit floating point value represented using single precision format is divided into 3 sections. The first bit in the single precision format is the Sign bit, the next 8 bits represents the Exponent, and the remaining 23 bits represents the Mantissa. The base of the exponent bit is 2 and thus the range of the exponent is -127 to 128.

Both the Divider and Multiplier uses a pipelined architecture to carry out the operation. The Taylor series is used to develop an algorithm for pipelining the divider which is explained in Section [3.2](#) of this paper while the Multiplier uses a Wallace Tree Algorithm. Additionally, a Look up Table and Multipliers are included in the Divider architecture. The Wallace Tree Algorithm enables pipelining the multiplier by calculating all the partial

products in the first clock cycle and then adding them using a set of half and full adders. Both of these algorithms are described in the later Section 3.2 and Section 4.1 of this paper. The goal of this project is build a Single precision floating point Divider and Multiplier using pipelined architecture where once the pipeline is filled, the divider and multiplier will generate an output at every clock cycle, thus reducing the latency.

1.1 Organization

Chapter 2 discusses different Divider Algorithms and provides background research. Chapter 3 presents the design and implementation of the Single Precision Floating point Dividers and the Architecture. Chapter 4 discusses the design and implementation of a Floating Point Multiplier using Wallace Tree Algorithm. Chapter 5 shows the tests and Result of the implementation of Single Precision Floating Point Divider and Multiplier and compares it's results implemented on different technologies. Finally, Chapter 6 discusses about the Possible future work and concludes the paper.

Chapter 2

Background Research

Dividers are an important part of today's Graphics Processor Units (GPU) and a good amount of work has been done by previous researchers in this field of study. Most of the Dividers fall into two categories namely Slow Dividers and Fast Dividers. Slow Divider's are the one's which are designed using the basic divider algorithm which is the shift and subtract algorithm discussed in the Section [2.1](#) of this chapter. While Fast Divider's can be designed using various algorithms like the Newton-Raphson Algorithm, Goldschmidt [\[6\]](#), etc. and pipelining stages as described in this chapter. A divider for handling complex number is described in [\[7\]](#). Different Dividers have been implented using Multiplication operations described in [\[8\]](#) and [\[9\]](#). Double precision floating point dividers have been described in [\[10\]](#) and [\[11\]](#). Dividers can be further categorized into Fixed-point or Floating point Dividers.

2.1 Shift and Subtract Divider

This is the simplest and basic divider designed by subtracting the divisor from the dividend and then comparing the result with the dividend. If the subtraction does not result into a negative number, the quotient is incremented by 1 and then the divisor is again subtracted from the dividend. This procedure is carried out till a negative result is achieved and then stored as a remainder [12]. The algorithm of this divider is shown in Figure 2.1. This algorithm is easy to implement and takes less hardware but it is extremely slow and thus cannot be used in today's fast GPU processors. Also this algorithm is very difficult to implement for floating point numbers. Due to these two major drawbacks, various different Algorithms were developed which are described in the Section 2.2 and Section 2.3 of this chapter.

2.2 Newton-Raphson Algorithm

Initially this method was developed in Numerical Analysis by Isaac Newton and Joseph Raphson, for finding the roots of a real-valued function. It was later developed for division operation's. Different Mathematical operations such as Multiplication and Subtraction are required in the Newton-Raphson's computational algorithm. At the start of the algorithm, the reciprocal of the Denominator is calculated and then it is multiplied with the Numerator to find the final Quotient. This multiplicative inverse of the denominator is calculated using iterative process and then the calculated multiplicative inverse is multiplied with the dividend to compute the final Quotient.

In the design mentioned in [13], the division algorithm is implemented using a 32 Bit floating point Multiplier and Subtractor Module. Scaling of the Divisor is carried out to achieve minimal and maximal relative error where the scaling is kept within the interval

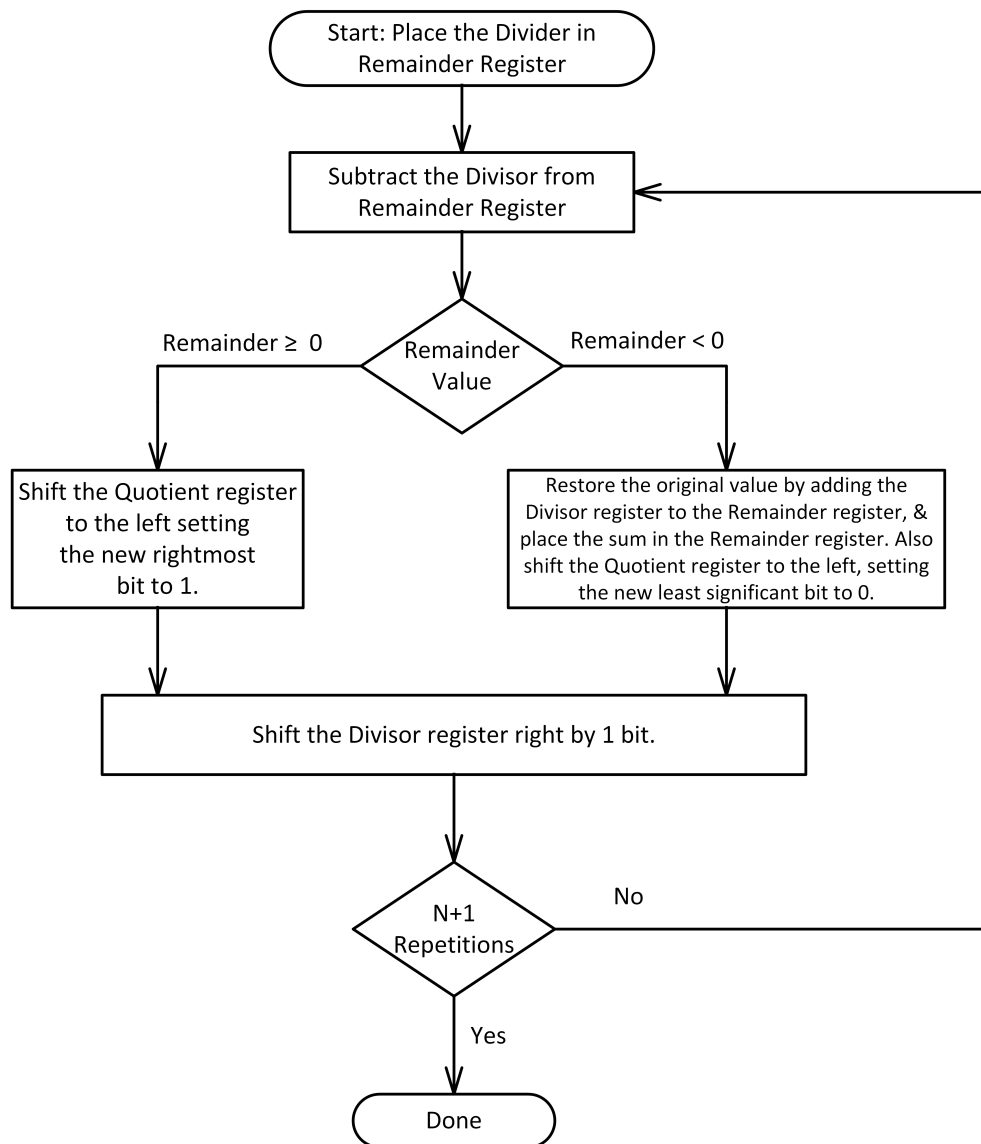


Figure 2.1: Basic Shift Divider[1]

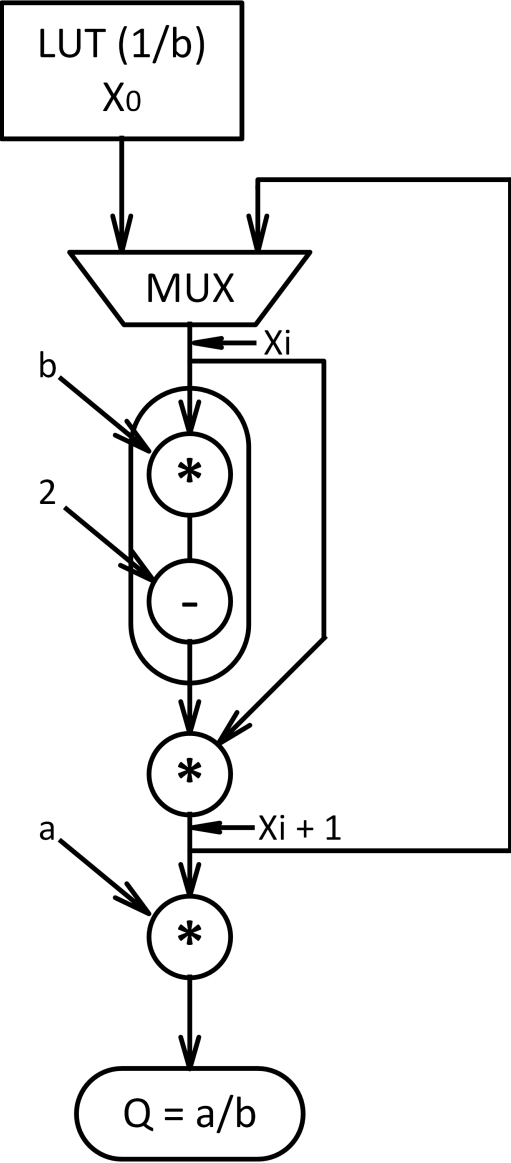


Figure 2.2: Newton - Raphson's Divider [2]

(0.5, 1). This scaling of the divisor is achieved by using shift operators. The algorithm is designed in such a way that to achieve higher accuracy, more iterations are required. The Newton-Raphson Algorithm converges faster for a single iteration and thus requires several Multiplier and Subtractor module's to compute continuous iterations [14]. Figure 2.2 shows the flowchart of a Floating Point Division using Newton_Raphson Algorithm. For a single iteration in this algorithm, two Multiplier and a Subtractor module are used. In [13] 3 iterations are computed in one cycle to get the required division result. The result can be improved by refining the multiplicative inverse.

The major disadvantage of the Newton_Raphson computational algorithm is computational failure if certain conditions of convergence are not met. This limits the use of this Algorithm as reliability of results being a major issue. Also the use of the multiplicative inverse results in the divider requiring a larger area compare to other algorithms [15]. Another disadvantage with the standard form of the Newton- Raphson divider is that the multiplications in the iteration are dependent to the previous stage and thus it must be performed serially. Hence two or more multiplications are required in each iteration [2].

2.3 Liddicoat and Flynn Divider

The Liddicoat and Flynn divider uses the first three terms of Newton and Raphson Divider [15]. The number of multiplication iterations is reduced in this divider and thus resulting in reduced latency [16]. The quotient can be directly expressed as a product of the dividend and the k^{th} order Newton-Raphson divider. The value of k decides the accuracy and the size of the Divider; the reater the value of k , the higher the accuracy and higher the complexity and area of the Divider [17]. This creates a trade-off between the size and the computational complexity of the algorithm.

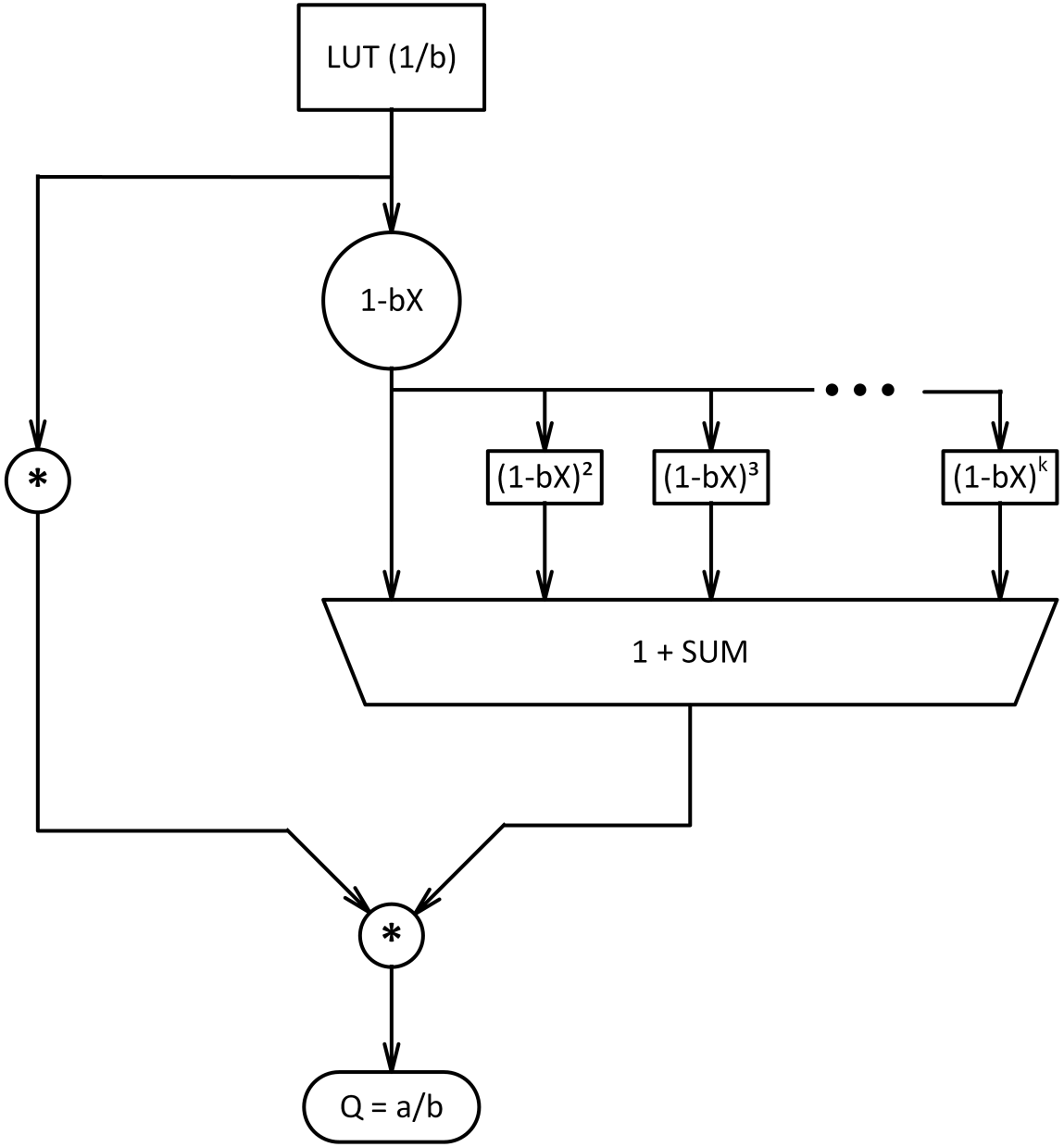


Figure 2.3: Liddicoat and Flynn Divider [2]

As seen from Figure 2.3, the value of $1/b$ is stored in a look up table. The selection of the size of the register b implies a trade-off between the lookup table size and error induced in computation. The next step is to express the quotient directly as the product of the dividend and the Newton-Raphson reciprocal approximation which is found by Eqn. 2.1.

$$aX(1 + (1 - bX)^1 + (1 - bX)^2 + \dots + (1 - bX)^k) \quad (2.1)$$

Here the value of X is the initial estimation of the dividend which is found in the look up table. This equation is achieved by using Binomial Expansion series from the Newton-Raphson Divider [13]. As the number of terms increases, the error in the quotient reduces resulting in increase in the complexity for implementation.

As only the MSB part of the dividend is stored in the Lookup table to reduce the size of it, the value of X generally contains an error due to approximation. The number of computation cycles required for an n -bit reciprocal stored in the Lookup Table containing only the MSB part required in a single iteration will be $\frac{n}{k+1}$ bits for a k^{th} order approximation. Also the number of bits of precision of X are increased approximately by a factor of $k + 1$ for a k^{th} order number[2], [18].

For divider shown in Figure 2.3, the divider unit can be fully pipelined by using two small multipliers, a powering unit for computing n^{th} order Newton_Raphson coefficient and a full multiplier to get the final product. The small multipliers are used to calculate the intermediate coefficients $(1 - bX)$, as X being calculated using the Lookup Table[2].

This divider architecture can be implemented using a fully pipelined version and thus can produce divide instruction per clock cycle. This divider utilizes the higher-order Newton-Raphson reciprocal approximation, this limits the accuracy to the number of coefficients used in the implementation. This is the major disadvantage of the Liddicoat and

Flynn Divider as the accuracy is limited to the number of Newton-Raphson divider and thus higher accuracy leads to increase in the complexity of the divider [15].

Chapter 3

Design and Algorithm

This chapter discusses the Design and Architecture of the Single Precision Floating point Divider using pipeline approach.

3.1 P. Hung's Divider using Taylor Series

P. Hung proposed a new high radix division algorithm which was based on Taylor Series expansion. With a slight modification in the Taylor Series, P. Hung showed a way to pipeline this algorithm. The previous algorithms described in chapter 2 considered each term of the Taylor series as a separate element and thus large look-up tables were needed to store the values of the coefficients thus making the algorithms more complicated [19]. Also reducing the size of the lookup table would make the results more error prone and thus had this major drawback.

P. Hung proposed an algorithm which would combine the first two terms of the Taylor Series and together, this would drastically reduce the size of the lookup table required to implement the algorithm and would provide more accurate results. As seen from Figure

3.1, the algorithm achieves a faster division by a getting the value from the lookup table and at the same time calculating $(y_h - y_l)$. This saves one clock cycle and thus reduces the latency of the division operation. This value of y_h and y_l is derived from the Dividend which is considered as Y which is described in section 3.1.1.

3.1.1 P. Hung's Algorithm

Consider the Dividend X and Divisor Y to be two $2m$ bit numbers between one and two. These two numbers can be defined by the following equation.

$$X = 1 + 2^{-1}X_1 + 2^{-2}X_2 + \dots + 2^{-(2m-1)}X_{2m-1} \quad (3.1)$$

$$Y = 1 + 2^{-1}Y_1 + 2^{-2}Y_2 + \dots + 2^{-(2m-1)}Y_{2m-1} \quad (3.2)$$

These two equations 3.1 3.2 have been derived from Taylor Series Expansion [3]. From the Equations 3.1 and 3.2, the divisor Y is separated into two parts called y_h and y_l where the y_h contains the MSB part of Y which is $(m + 1)$ bits wide while y_l contains the LSB part of Y which is $(m - 1)$ bits wide. Thus the division equation can be written as

$$\frac{X}{Y} = \frac{X}{(Y_h + Y_l)} \quad (3.3)$$

$$\frac{X(Y_h - Y_l)}{(Y_h + Y_l)(Y_h - Y_l)} \quad (3.4)$$

$$\frac{X(Y_h - Y_l)}{Y_h^2} \quad (3.5)$$

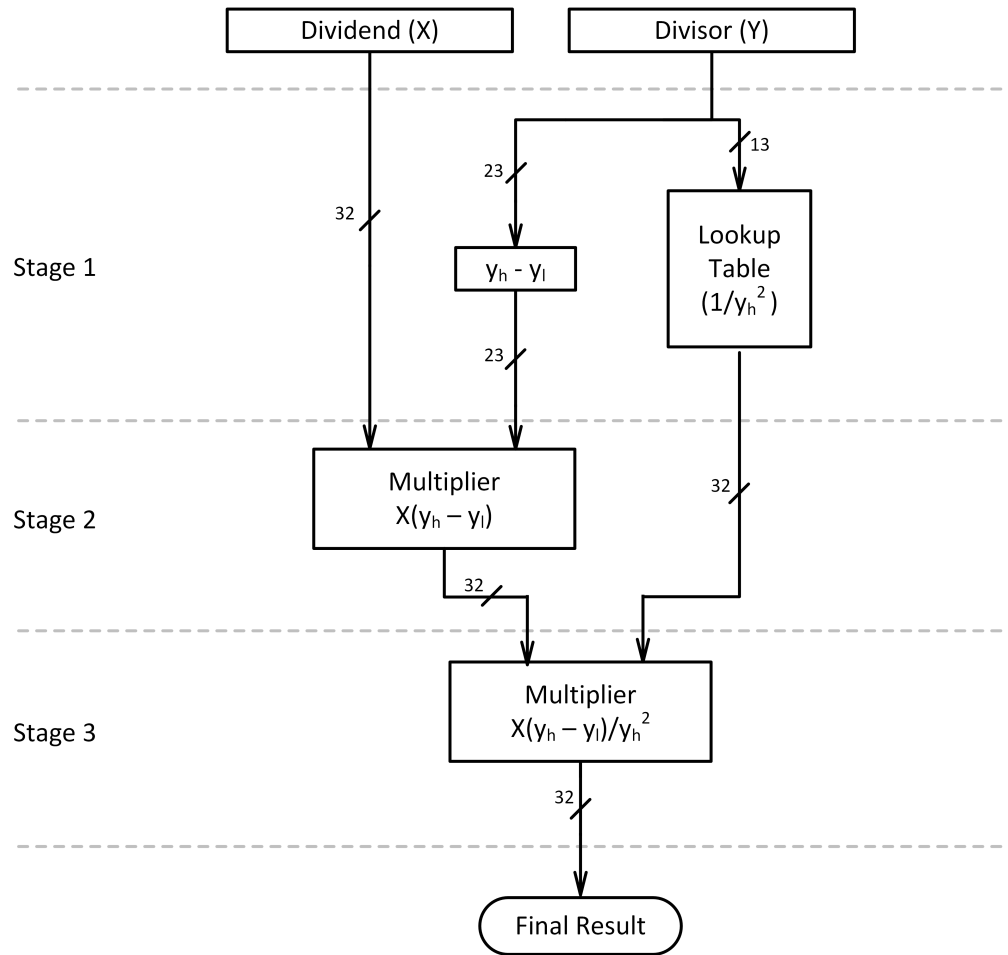


Figure 3.1: P. Hung's Divider Algorithm using Taylor Series [3]

Equation 3.5 is the final equation for Hung's divider. According to Equation 3.5, the division operation can be performed by mean's of multiplying the Dividend X with $(Y_h - Y_l)$ and also with the $1/(Y_h)^2$. As seen in Figure 3.1, the divider can be thus implemented using 3 stages of Pipeline. In the first stage the value of $(Y_h - Y_l)$ is calculated. Also the value of $1/(Y_h)^2$ is taken from the lookup table. Using a multiplier, X is then multiplied with $(Y_h - Y_l)$ term in the second stage of the pipeline. In the 3rd stage the result of the second stage is multiplied using a multiplier with $1/(Y_h)^2$ term to get the final result.

3.1.2 Error Analysis of P. Hung's Algorithm

The term $1/(Y_h)^2$ is stored in a Look up table and thus plays an important role in the overall size of the Divider. The divider size can be reduced by decreasing the bit width of Y_h . For every one bit decrease in the bit width of Y_h , the lookup table size reduces by 2 times and vice-versa. But this decrease in size of the lookup table comes at a cost of increase in the error of the final result [20]. Apart from this, there are 3 other sources of error in this Division Algorithm namely Taylor Series Approximation error, Rounding error due to first multiplication, and Rounding error due to Second multiplication [3]. The total error can be expressed using the following equation 3.6.

$$E_T = E_L + E_{TS} + E_{M1} + E_{M2} \quad (3.6)$$

Where,

E_T = Total Error

E_L = Lookup table rounding error

E_{TS} = Taylor Series Approximation Error

E_{M1} = Rounding error due to first Multiplication

E_{M2} = Rounding error due to second Multiplication

For minimizing the error, the divider needs to be carefully designed such that $E_L \leq 0$, $E_{TS} \leq 0$, $E_{M1} \leq 0$ and $E_{M2} \geq 0$ [3]. To achieve this, the result of the first multiplication should be truncated to $2m + 2$ bits of MSB thus resulting in value of Y_h should also be $2m + 2$ bits. The result of the second multiplication should be truncated to $2m$ bits of MSB which will be the final result. The total error generated by all of these factors is less than 1ulp, out of which the lookup table and first multiplication generate an error of less than 1/4 Unit of Least Precision (ulp) while the second multiplication generates an error

of less than 1ulp.

3.2 Single Precision Floating Point Divider Algorithm

The division algorithm mentioned in Section 3.1 requires a lookup table whose size can be decreased by approximation. But even after this reduction, the size of the lookup table is still 13KB for a single precision division [15]. Thus, this size of the lookup table constitutes a larger area compare to that of iterative dividers. Also it is impossible to implement this algorithm for a double precision number considering the large amount of lookup table size. Also as discussed in Section 3.1.2, the error induced in the Divider of Double Precision would greatly increased and thus this algorithm proves to be unsatisfactory for a Double Precision Floating point Division.

This paper shows an Algorithm described in [15] where the size of the Lookup Table used in this Paper is just 2KB compare to the large size of Lookup table used in [3]. The algorithm mentioned in [3] is also slightly modified to reduce the error. A course and a fine quotient is found and then added together to get the final result. This algorithm is explained in detail in the later sections of this chapter. Section 3.2.1 describes the IEEE-754 standard for Single Precision Floating Point number. Section 3.2.2 describes the Algorithm and Architecture used in this project while Section 3.2.3 shows the Error Analysis and Error induced due to this Algorithm.

3.2.1 IEEE-754 Standard for Floating Point Numbers

The IEEE-754 Standard was developed specially for Computers to recognize and perform operations on Floating point numbers developed by the Institute of Electrical and Electronics Engineers in 1985 which was later modified in June 2008. It provides a standardize

method for performing computation in floating point point numbers. The major advantage of this standard is to be able perform operations on floating point numbers independent of it being on a software or hardware platform or even combination of the two.

The IEEE 754 standard allows the user to specify the floating point number into various different formats such as Single Precision, Double Precision, Extended and Extendable Precision according to user requirements. This standard includes formats to define both Binary and decimal floating point formats, performing Arithmetic operations on them, conversion of Binary Floating point to Decimal Floating point and vice-versa and also Exception handling for numbers who comes in the category of Not A Number (NaN) [21] defined in Section 3.2.1.2. The floating point formats in Single and Double Precision format is discussed in the below Section 3.2.1.1

3.2.1.1 Floating Point Format

IEEE 754 standard allows the user to define a floating point number into various different formats such as Single Precision, Double Precision, Extended and Extendable Precision. In this section, Single and Double Precision format for representing floating point numbers is discussed. Both of the these formats are divided in three parts namely Sign Bit, Exponent Bit and Mantissa or Significand Bit, the difference being in the size of the Single Precision and Double Precision format shown in Figure 3.2 and Figure 3.3

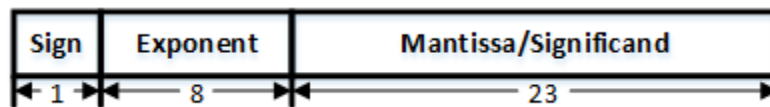


Figure 3.2: Single Precision Floating Point Format[4]

. The Single Precision format consists of 32 bits whereas Double Precision format consists of 64 bit where the MSB 1 bit defines the Sign of the number in both the format.

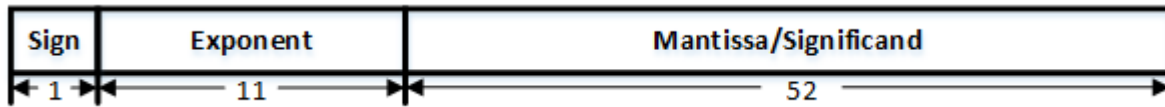


Figure 3.3: Double Precision Floating Point Representation[4]

The Exponent part in the Single precision format is stored in excess 127 format where the range of the exponent part can be between 128 to -127. Similarly for the double precision format, the exponent part is represented in excess 2048 format and the range of it is between 1023 to -1022. Thus a very small number can be defined using both of these formats. The Mantissa/Significand part is a bit tricky in both of these formats. The mantissa is always stored in normalized form which means 1 is considered to be always present in the MSB part of the Mantissa also called as implicit 1 in the MSB. As this 1 is always considered to be present in the MSB this 1 in the MSB is never stored in Mantissa part of both Single and Double Precision format but it is always used if any operation considering it to be present. It can be explained by the following example.

Eg. Number 4.5 can be written in Floating point Binary as

$$4.5 = 100.1$$

The result in the normalised form is represented as

$$4.5 = 1.001 \times 2^2$$

To store this number in the Single Precision form, the Mantissa is stored as 001 while the exponent is incremented by 2 to take care of the normalized form. Therefore 4.5 in single precision format is stored as 0x4090_0000. Due to this normalization, storing the Mantissa part becomes a slightly complex operation due to the implicit hidden 1 in the Mantissa Part. If the value of the Mantissa part is less than 1.0, then a left shift operation is performed till the MSB bit is 1. The number of times this left shift operation is performed, the Exponent bit is decreased by the same amount. Similarly, if the number is greater than

2.0, then a right shift operation is performed till the value is than 2.0 and the exponent bit is increase by the same amount [22]. This operation is applicable to both the Single and Double Precision format.

Processing on the Double Precision Floating Point format is slower than it's counter part Single Precision Floating Point form due to its size. Thus Single Precision format is used in most of the processors. Majority of the times, the Double Precision format is used for Parallel computing in the GPU's.

3.2.1.2 Not A Number (NaN)

IEEE 754 standard has an inbuilt way to define not defined numbers. Operations such as a number divided by 0 will result in infinity and thus result in exception case of number overflow. Also operations such as taking a square root of a negative number will result in an Imaginary number which will trigger the exception handling of this format number. The IEEE 754 standard defines such exception numbers as Not a Number (NaN). It should be taken into account that NaN numbers are not equal to infinity, it is just a technique to represent exceptional cases in operation [21], [4]. IEEE - 754 format supports two types of NaN namely Signaling and Quiet in all floating point operations [21].

3.2.2 Algorithm and Architecture

Equation 3.5 can be considered as the Course Coefficient of the Quotient. As discussed in Section 3.1, Hung's divider induces an error in the final result. This error can be reduced by introducing adding Course and Fine Coefficients of the Quotients. The course coefficient

from Equation 3.5 can be defined as

$$\tilde{Y} = \frac{X(Y_h - Y_l)}{Y_h^2} \quad (3.7)$$

Where \tilde{Y} is defined as the course coefficient.

From [15], the subdividend \tilde{X} can be defined as,

$$\begin{aligned} \tilde{X} &\approx X - Y \cdot \tilde{Q} \\ &= X - Y \cdot \frac{X(Y_h - Y_l)}{Y_h^2} \\ &= X \left(1 - Y \cdot \frac{(Y_h - Y_l)}{Y_h^2} \right) \\ &= X \left(1 - \frac{(Y_h + Y_l)(Y_h - Y_l)}{Y_h^2} \right) \\ &= X \left(1 - \frac{Y_h^2 - Y_l^2}{Y_h^2} \right) \\ &= X \left(\frac{Y_h^2 - Y_h^2 + Y_l^2}{Y_h^2} \right) \\ \therefore \tilde{X} &= X \left(\frac{Y_l^2}{Y_h^2} \right) \end{aligned} \quad (3.8)$$

This Subdividend \tilde{X} can be utilized to find the fine coefficient $\tilde{\tilde{Q}}$. This can be found

using the value of \widetilde{X} from equation 3.8 in equation 3.7 which would give us.

$$\widetilde{\widetilde{Q}} = \frac{\widetilde{X}(Y_h - Y_l)}{Y_h^2} \quad (3.9)$$

According to [15], the Final Quotient can be derived by adding Equation 3.7 and 3.9,

$$\begin{aligned} Q &= \widetilde{Q} + \widetilde{\widetilde{Q}} \\ &= \frac{X(Y_h - Y_l)}{Y_h^2} + \frac{\widetilde{X}(Y_h - Y_l)}{Y_h^2} \\ &= \frac{(X + \widetilde{X})(Y_h - Y_l)}{Y_h^2} \end{aligned} \quad (3.10)$$

$$\begin{aligned} &= (X + \widetilde{X})A \\ &= (X + X - Y \cdot \widetilde{Q})A \\ &= (2X - Y \cdot AX)A \end{aligned}$$

$$= (2 - AY)AX \quad (3.11)$$

where,

$$A = \frac{(Y_h - Y_l)}{Y_h^2} \quad (3.12)$$

Equation 3.11 refers to the final equation of the Single Precision Floating Point Divider. Equation 3.11 can also be implemented for Double Precision floating point division by increasing the size of the Multiplier and Lookup table[23]. This algorithm can be implemented using the Pipeline Architecture shown in Figure 3.4.

In the first stage, the dividend Y is separated into two terms Y_h and Y_l where the Y_h is 12 bits MSB, while Y_l is 11 bits LSB. Y_h is then concatenated with the implicit 1 of the Mantissa in Floating Point Representation. The value of Y_h thus becomes 13 bits which is then subtracted with Y_l .

The lookup table is constructed in the form of ROM implementation and the input Y_h is considered to be the address of the memory location. The memory location the value of $1/(Y_h)^2$ is stored in Single precision floating point format. This storage in the single precision format is important because the equation $1/(Y_h)^2$ would result in a floating point number. Thus it needs to be stored in a standardized format so that the resultant value from the Lookup table can be easily utilized in the later stages of the Pipeline. At the same time, value of $(Y_h - Y_l)$ is also calculated. As mentioned above, Y_h is concatenated with 1 at the MSB, therefore the result of this subtraction is never a negative number. This value is also converted into Single Precision Floating point format to simplify the further operations.

These results achieved in first stage of pipeline are then multiplied with each other to obtain the value of A described by Equation 3.12. A 32 bit floating point multiplier is used to perform this operation. As discussed in Section 3.2.1, special care needs to be taken for floating point multiplication. The 32 bit number in Single Precision format is first separated into its Sign, Exponent and Mantissa bits. This number is then checked with all the NaN conditions and then if this test passes then the Mantissa is unpacked from its Normalized form [22]. The 23 bit Mantissa is converted into a 24 bit number and then multiplication

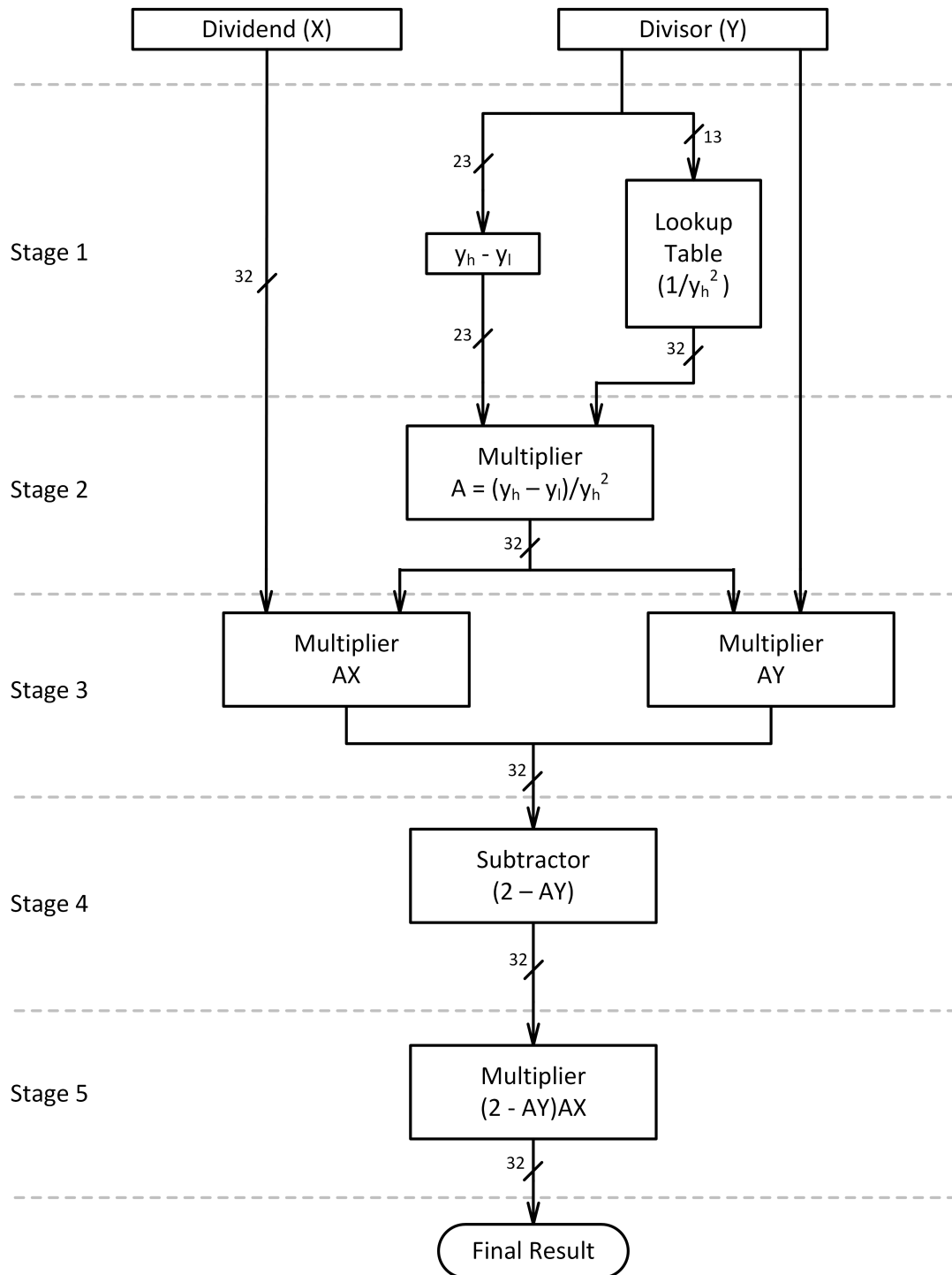


Figure 3.4: Architecture of the Divider

of the Mantissa is carried out [5]. The exponent bit of the two inputs are added while an Ex-OR operation is carried out with the sign bit of both inputs to the Multiplier [24]. The architecture of the Multiplier used in this operation is described in Chapter 4. The result of the multiplication operation is truncated to 24 bits and then a packing operation is carried out on to Normalize the Mantissa. Finally, the Sign, Exponent and Mantissa bits are then concatenated to get the final result of Floating point multiplication. The value of A achieved in the second stage is multiplied with the Dividend X and Divisor Y to get the product AX and AY . Similar multiplication operation is carried out in the third stage as to the second stage.

In the fourth stage of Pipeline, the $(2 - AY)$ operation is carried out using a floating point subtracter module. Similar to multiplication operation, the mantissa needs to be unpacked for Subtraction too. But here, the exponent of both the inputs first needs to be made equal and the Mantissa bits are adjusted accordingly. After unpacking the subtraction operation is carried out on the Mantissa bits. The same algorithm can be used for addition by just Adding the Mantissa bits instead of Subtracting. The Sign, Exponent and Mantissa bits are then concatenated to get the result of Pipeline Stage four. This result is then multiplied with the product of AX using the same multiplier algorithm explained above to get the final result of Division operation.

3.2.3 Error Analysis

Similar to Section 3.1.2, even this algorithm induces an error. There are three types of error induced in the algorithm which are error due to the lookup table, rounding error in Multiplication operation and rounding error in Subtraction operation. Four Floating point multipliers are used in the overall algorithm and the total error in the algorithm is

as follows,

$$E_T = E_{LT} + E_S + E_{m1} + E_{m2} + E_{m3} + E_{m4} \quad (3.13)$$

Where,

E_T = Total Error

E_{LT} = Lookup table rounding error

E_S = Rounding error due to Subtraction

E_{M1} = Rounding error due to first Multiplication

E_{M2} = Rounding error due to second Multiplication

E_{m3} = Rounding error due to third Multiplication

E_{m4} = Rounding error due to fourth Multiplication

The lookup table error is due to limiting the storage of the $1/(Y_h)^2$ term. This is done to reduce the size of the lookup table. This reduction in the bit width of the Y results in the Lookup table error. The multipliers used in this algorithm are 24 bit multipliers which results in a 48 bit result. Thus the result of this multiplication operation is truncated to 24 bits MSB which results in the rounding error. All the errors can be minimized similar to the method discuss in Section 3.1.2. In [15] an error due to limitation of entries in the lookup table is also introduced as they restrict the number of entries to reduce the size of the lookup table which is not the case in the algorithm used in this paper.

Chapter 4

Wallace Tree Multiplier

Similar to Division, Multiplication is also an important arithmetic operation in today's processors. The basic multiplier consists of Adders, Shifters and AND gate to perform the multiplication shown in figure 4.1. A basic 4 bit multiplier is shown in the Figure 4.1 which results in an 8bit product. PPO, PP1, PP2 and PP3 are the partial products generated using the AND gate in the multiplication operation. A sum of this partial product is calculate using ADDER to generate the final result of Multiplication. The number of partial products generated increases with respect to the Multiplier's bit size. For a 16 bit adder, the 256 partial products are generated. This number can be reduce by using Booth's Encoding which is a technique to reduce the number of Partial products generated during a multiplication operation [24].

Booth's Algorithm is one of the most commonly used algorithm for Multiplications operation used for both signed and unsigned numbers. It uses Shift operation which are faster than Adders, to perform multiplication. A high speed pipelined divider in discussed in [24] where they use booth encoding to reduce the number of partial products. Figure 4.2 shows Booth's Multiplier Algorithm for a 32 bit Multiplier. It checks the 2 bit's of

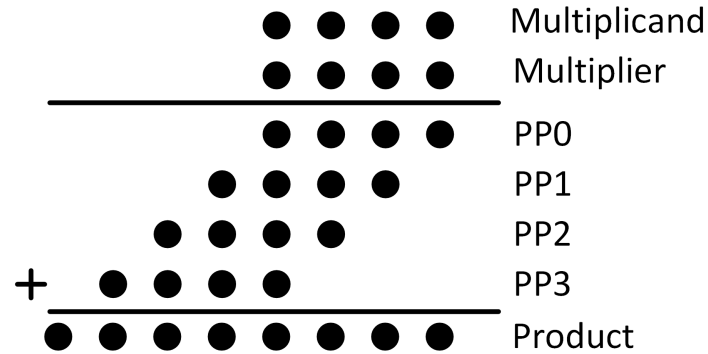


Figure 4.1: Basic Multiplier [5]

the multiplicand and does the operation accordingly. As seen from Figure 4.2 it performs No Operation if these 2 bits are 00 or 11 while it performs an Addition or Subtraction operation depending on these two bits.

Booth encoding is used in [5] where the generated partial products are later added using Carry Look Ahead Adder. Booth's encoding is a technique used to reduce the number of Partial Products generated which are the main agent of time required for Multiplication operation. Booth's encoding can reduce these number of Partial products generated to almost half by using Radix-4 technique.

As seen from Figure 4.3 a Carry Look Ahead Adder is used to calculate the sum of the partial product. This CLA's have a huge drawback of propagation delay due to the fact that every addition operation has to wait for the previous Carry bit to be generated creating a huge delay in operation. As the CLA is used in every stage, the propagation delay induced in this Multiplier would huge and may not meet the timing requirement. To reduce the propagation delay induced due to this CLA adders, a Wallace Tree Algorithm described in Section 4.1 can be used.

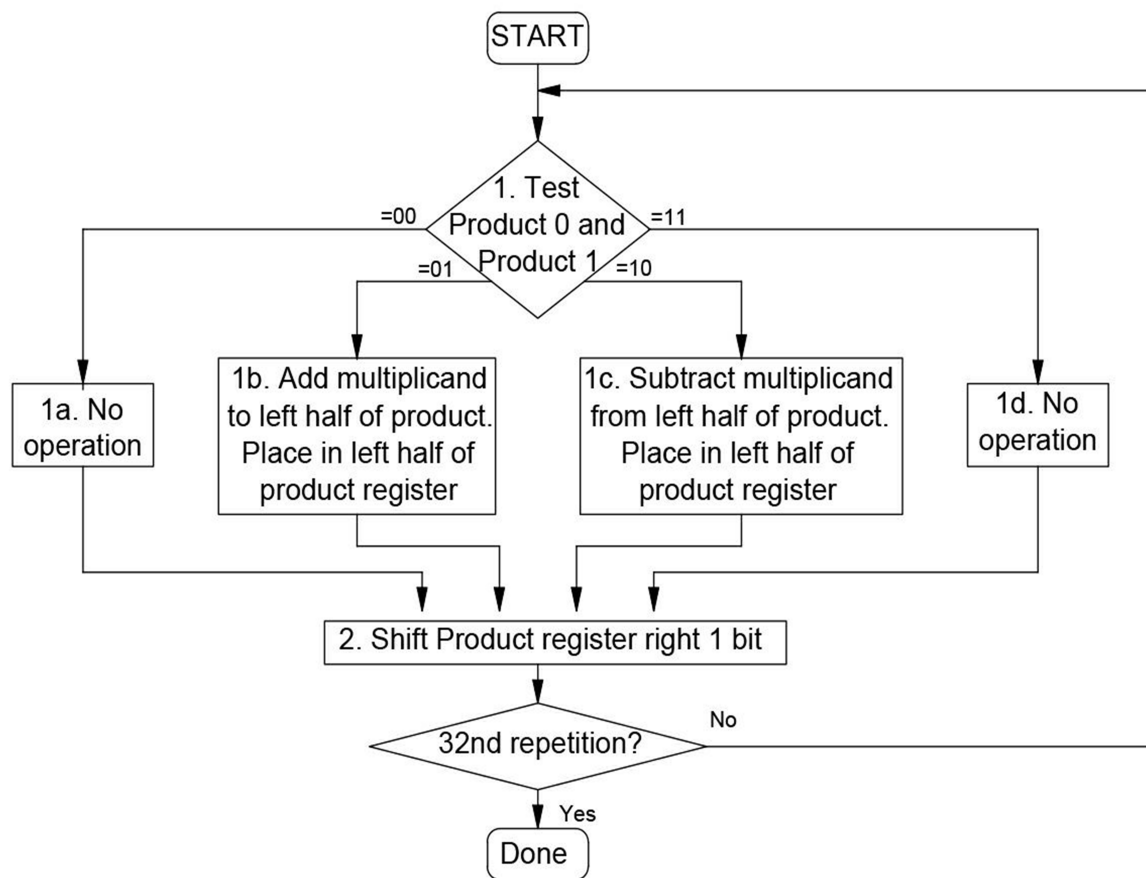


Figure 4.2: Booth's Algorithm

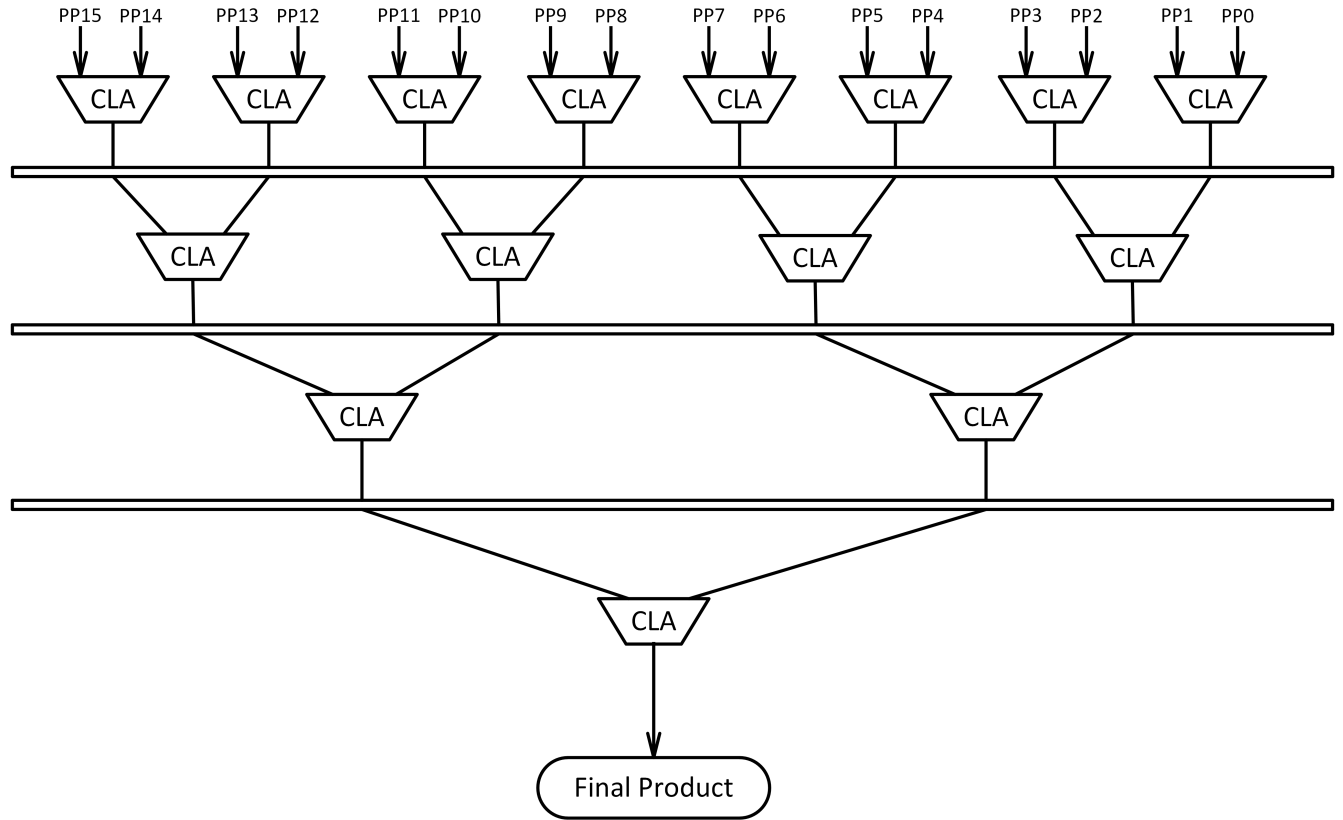


Figure 4.3: 8 Bit Multiplier Using CLA Adder [5]

4.1 Algorithm

The major advantage Wallace Tree Algorithm provides compared to the traditional CLA adders in the Multiplier is that the Adder operation does not have to wait for the previous stage Carry. It only considers the previous stage generated Carry at a later stage instead of waiting it during every Adder operation. This reduces the propagation delay drastically as we increase the size of the multiplier.

Figure 4.4 shows a Wallace Tree algorithm for an 8bit Multiplier. In Wallace Tree Algorithm, all the Partial products of the Multiplication are calculated in the first stage. These partial products are calculated using an AND gate where each bit of Multiplicand is ANDed with every bit of the Multiplier. Sum of partial products is calculated using Half

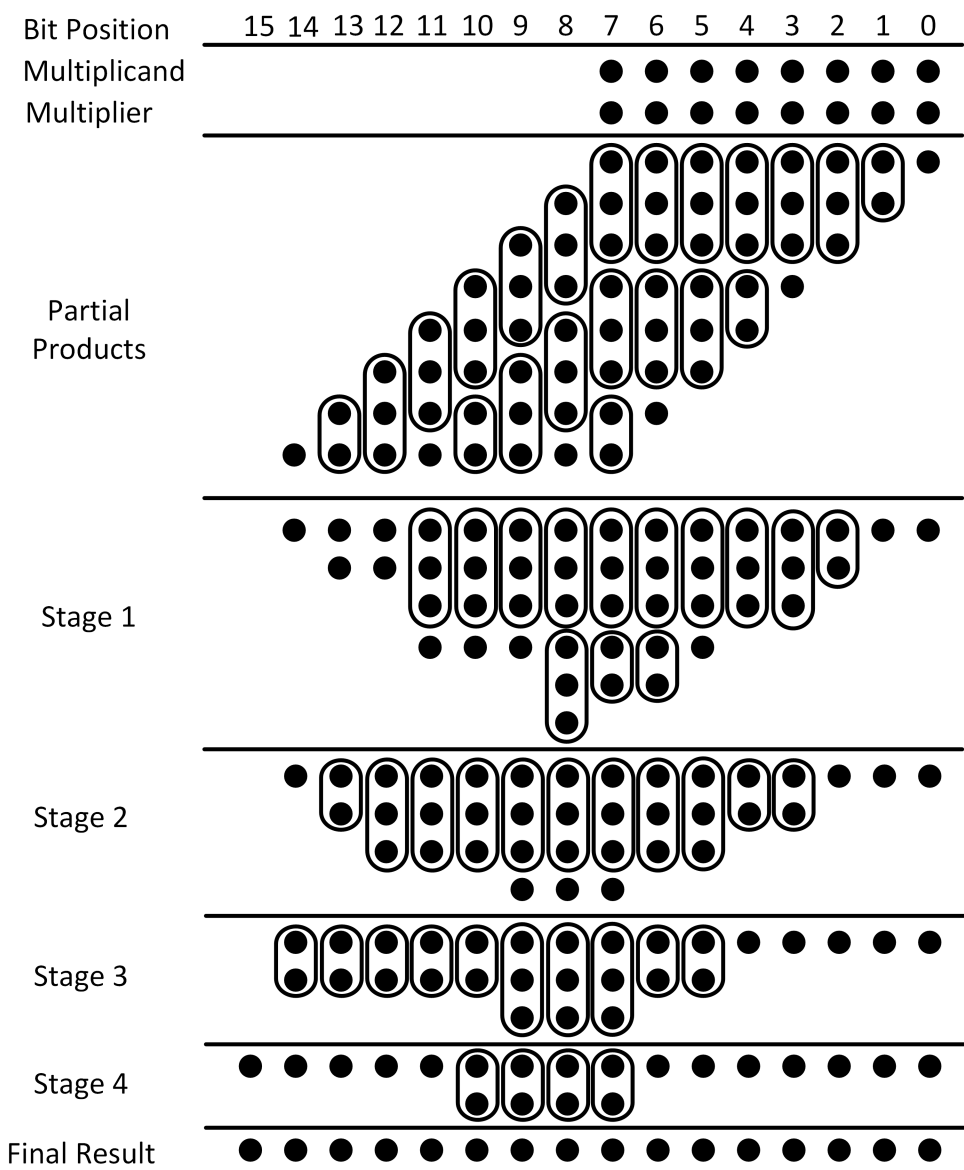


Figure 4.4: 8 Bit Wallace Tree Multiplier

and Full Adders according to the number of Partial products in each column. Whenever an Addition operation is performed in a column, the generated carry is placed in the neighboring column similar to CLA adder but the addition in the neighboring column is independent of this generated carry in that stage [25]. In the next stage, this generated carry is used in the addition operation and thus the addition operations are not restricted because of the generated carry. As seen from figure 4.4, addition of the partial products is carried out in different stages, thus a pipeline architecture can be implemented to on Wallace tree multiplier.

An entirely combinational circuit is used to implement the Wallace Tree Multiplier in this project. All the partial products were calculated by using AND gate between every bit of Multiplicand and Multiplier. Approximately a 576 partial products were generated in the first stage on which addition operation was carried out in 10 stages. Half of the bits of the final results are generated by the 8th stage of the addition. This is achieved by adding according to the remaining partial products as the algorithm progresses. This further reduces the propagation delay as only half of the final result needs to be calculated in the last stage, thus resulting in a CLA adder only for half of the bits in the final stage.

Chapter 5

Results

This chapter discusses the results from the implementation of the Single Precision Divider and Multiplier.

5.1 Floating Point Divider

The Single Precision Floating Point Divider was implemented using the Architecture described in Chapter 3.2.2. It was synthesized on different technologies using two pass synthesis strategy: RTL logic synthesis followed by Design for Test (DFT) synthesis. A full scan methodology was used in the synthesis for test structure insertion, which inserts scan chains throughout the design for testing. A clock of 50Mhz was used during the testing of the entire design.

5.2 Floating Point Multiplier

As discussed in Chapter 4, the Multiplier was implemented using a Wallace Tree Algorithm. Similar to the Floating point Divider, the Multiplier was also Synthesized on a different

Table 5.1: Area, Power, Timing, and DFT Coverage of Divider

		32nm	65nm	180nm
Area	Sequential Area	23901	31325	250607
	Combinational Area	84598	80222	563934
	Buf/Inv Area	4982	4060	15750
	Total Area	145245	111548	814542
Power	Internal Power	1.533E3 μ W	1.99 mW	14.4372 mW
	Switching Power	65.6268 μ W	0.1134 mW	2.3339 mW
	Leakage Power	1.025E10pW	5.2629e03 nW	3.807E3 nW
	Total Power	1.167E5 μ W	2.1161 mW	16.7749 mW
Timing	Slack	14.845ns	14.065ns	14.392ns
DFT Coverage		98.38%	99.54%	99.57%
Latency (Clock Cycles)		41	41	41

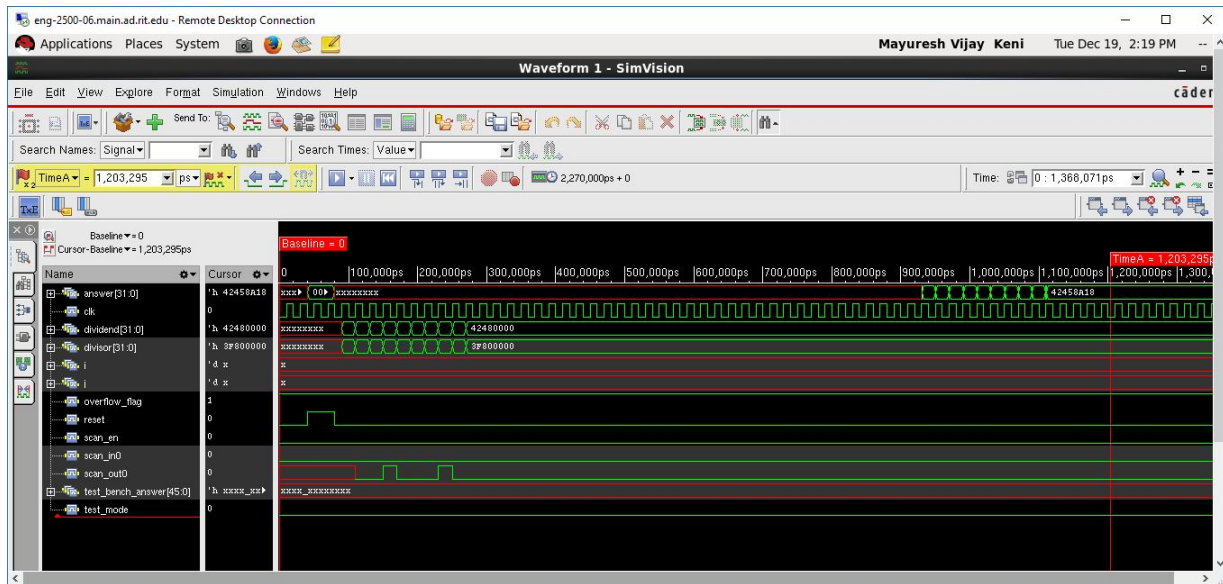


Figure 5.1: 32nm Divider Result

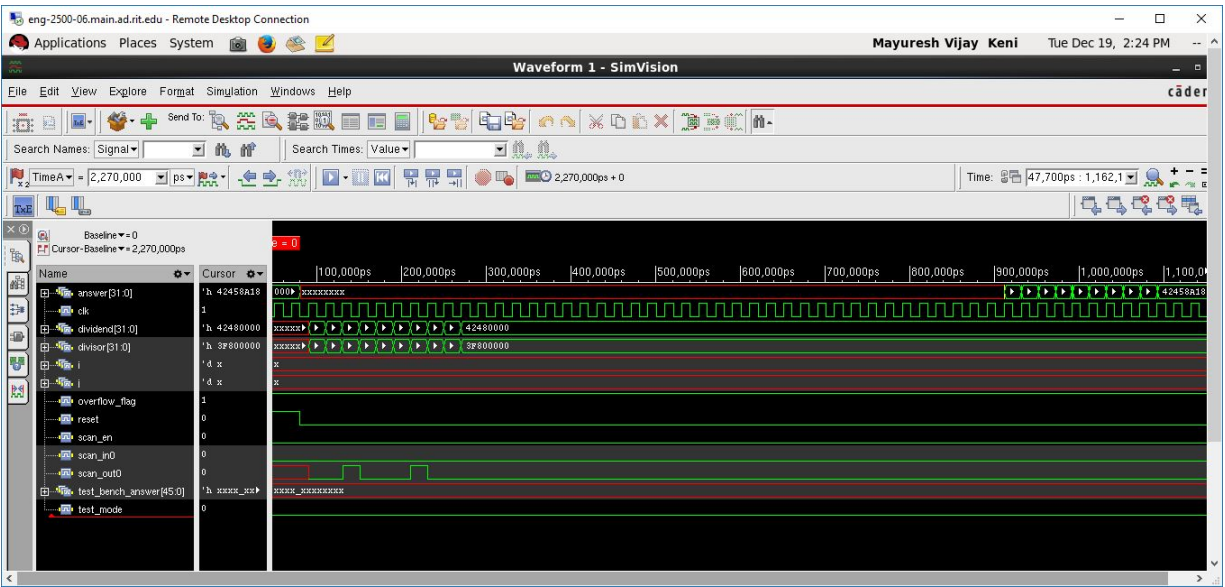


Figure 5.2: 65nm Divider Result

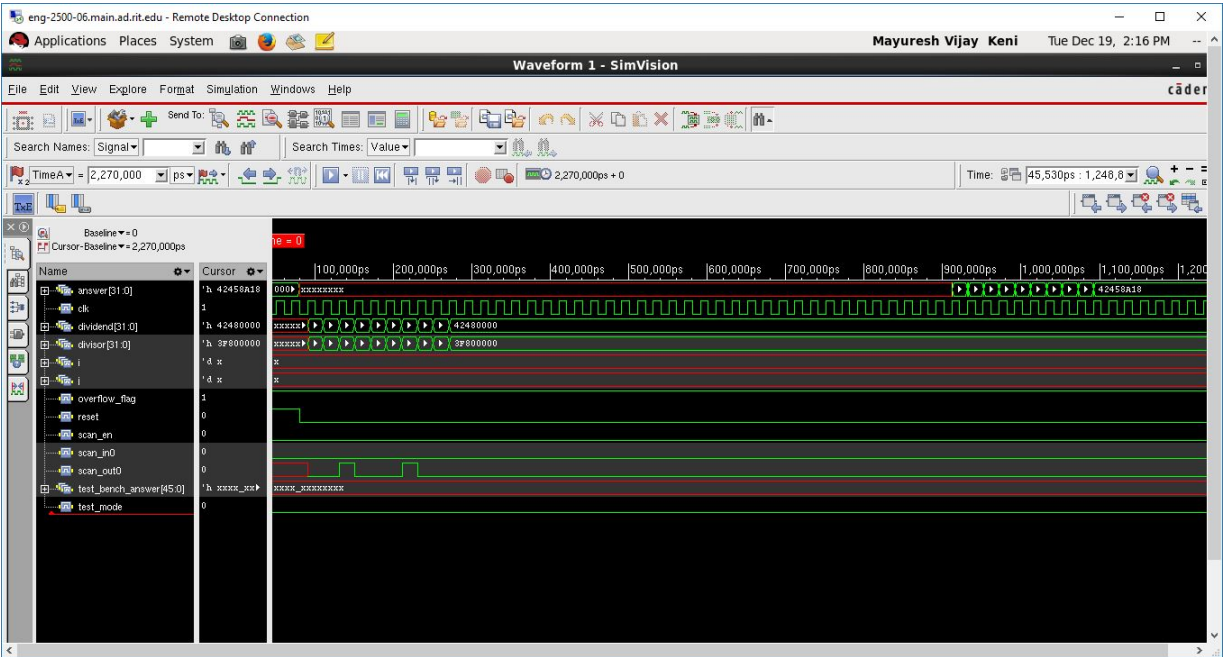


Figure 5.3: 180nm Divider Result

Table 5.2: Area, Power, Timing, and DFT Coverage of Multiplier

		32nm	65nm	180nm
Area	Sequential Area	2855	3507	28866
	Combinational Area	12178	11741	83991
	Buf/Inv Area	637	416	2125
	Total Area	19370	15249	112858
Power	Internal Power	197.876 μ W	0.2797 mW	2.169 mW
	Switching Power	19.6011 μ W	2.877E-02 mW	0.5953 mW
	Leakage Power	1.4197E09 pW	668.708 nW	591.9263 nW
	Total Power	1.6372E03 μ W	0.3092 mW	2.7649 mW
Timing	Slack	14.4904ns	14.923ns	16.5276ns
DFT Coverage		99.44%	99.42%	99.33%
Latency (Clock Cycles)		9	9	9

technologies using two different synthesis options, RTL logic synthesis and DFT Synthesis with a full scan methodology.

Chapter 6

Conclusions

This chapter discusses conclusions from this project as well as future work that could be completed.

6.1 Conclusions

This paper describes implementation of a Floating point Divider and Multiplier which was benchmarked on different technologies. A standard 31 stage pipeline was implemented on the Divider while a 9 stage Pipeline was implemented on the Multiplier. The goal of this project was to implement a Divider and Multiplier using pipeline architecture so that once the pipeline is filled, there will be an output at the end of every clock cycle.

6.2 Future Work

The multiplier and the divider can always be improved and optimized. This paper discusses about the implementation of a Single Precision Multiplier and Divider. In future the same algorithm can be implemented on a Double Precision floating point number. In this project,

the Multiplier has a latency of 9 clock cycles while the Divider has a latency of 31 clock cycles. In future, work can be done to reduce the latency of both of them. The number of partial products generated in the multiplier can be reduced by using Booth's Encoding.

References

- [1] D. B. Najib Ghatte, Shilpa Patil, “Single precision floating point division,” in *Fifth IRF International Conference*, vol. 13, 2014, pp. 294–298.
- [2] A. A. Liddicoat and M. J. Flynn, “High-performance floating point divider,” in *Digital Systems Design, 2001. Proceedings. Euromicro Symposium*, 2001.
- [3] P. H. H. F. O. M. M. J. Flynn, “Fast division algorithm with a small lookup table,” in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference*, 1999, pp. 137–150.
- [4] B. Parhami, *Computer Arithmetic*. Oxford University Press, 1999, vol. 42.
- [5] P. D. Smruti Bokade, “Cla based 32-bit signed pipelined multiplier,” in *Communication and Signal Processing (ICCSP), 2016 International Conference*, 2016.
- [6] G. E. P.-M. W. E.Ferguson, “A parametric error analysis of goldschmidt’s division algorithm,” in *Journal of Computer and System Sciences*, vol. 70 (2005) 118-139, Aug. 2014.
- [7] L. L. Dong Wang, Pengju Ren, “A high-throughput fixed-point complex divider for fpga,” in *IEICE Electronics Express*, vol. 10, no. 4, 2013.

-
- [8] P. MalÅk, “High throughput floating-point dividers implemented in fpga,” in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2015 IEEE 18th International Symposium*, 2015.
 - [9] I. Rust and T. G. Noll, “A radix-4 single-precision floating point divider based on digit set interleaving,” in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium*, 2010.
 - [10] X. Fang and M. Leeser, “Vendor agnostic, high performance, double precision floating point division for fpgas,” in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, 2013.
 - [11] V. S. R. D. M. V. H. Neto, “Multiplier-based double precision floating point divider according to the ieee-754 standard,” in *Springer-Verlag Berlin Heidelberg*, vol. 262-267, 2008, 2008.
 - [12] S. O. . M. Flynn, “Minimizing the complexity of srt tables,” in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, 1998, pp. 141–149.
 - [13] N. Singh and T. N. Sasamal, “Design and synthesis of single precision floating point division based on newton-raphson algorithm on fpga,” in *4th International Conference on Advancements in Engineering & Technology (ICAET-2016)*, 2016.
 - [14] W. G. . E. Swartzlander, “Fault-tolerant newton-raphson and goldschmidt dividers using time shared tmr,” in *IEEE Transactions on Computers*, vol. 49, 2000, pp. 588–595.
 - [15] J.-C. J. . W.-C. P. . W. J. . T.-D. H. . M.-K. Lee, “A cost-effective pipelined divider with a small lookup table,” in *IEEE Transactions on Computers*, vol. 53, no. 4, Apr. 2004, pp. 490–496.

-
- [16] T. J. K. Taek Jun Kwon, "Floating-point division and square root using a taylor-series expansion algorithm," in *Microelectronics Journal*, vol. 40, Apr. 2009, pp. 1601–1605.
- [17] A. L. . M. Flynn, "Parallel square and cube computations," in *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference*, vol. 24, 2002, pp. 1645–1667.
- [18] T.-J. K. . J. Draper, "Floating-point division and square root implementation using a taylor-series expansion algorithm with reduced look-up tables," in *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium*, 2008.
- [19] T.-J. K. J.-S. M. J. S. J. Draper, "A 0.18ÎEm implementation of a floating-point unit for a processing-in-memory system," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium*, 2004.
- [20] J.-C. J. W. J. H.-J. W. S.-H. K. W.-C. P. M.-K. L. T. don Han;, "A new pipelined divider with a small lookup table," in *ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference*, 2002.
- [21] I. C. S. M. S. C. I. of Electrical and E. E. I.-S. S. Board., *IEEE Standard for Floating-Point Arithmetic*, revision of ieee std 754-1985 ed. Institute of Electrical and Electronics Engineers, 2008.
- [22] J. L. H. Patterson, D. A., *Computer Organisation and Design. The Hardware/Software Interface*, 5th ed. Morgan Kaufmann, 2013.
- [23] S. K. N. Sandeep B. Singh, Jayanta Biswas, "A cost effective pipelined divider for double precision floating point number," in *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference*, Dec. 2006.

-
- [24] A. B. Qingzheng LI, Guixuan LIANG, “A high-speed 32-bit signed/unsigned pipelined multiplier,” in *Fifth IEEE International Symposium on Electronic Design, Test & Applications*, 2010.
- [25] R. D. K. A. A. S. V. P. Jayakrishnan, “Implementation of pipelined booth encoded wallace tree multiplier architecture,” in *International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, 2013.

Appendix I

Source Code

I.1 Floating Point Divider

```
1 module mvk_div (
2     reset ,
3     clk ,
4     scan_in0 ,
5     scan_en ,
6     test_mode ,
7     dividend ,
8     divisor ,
9     answer ,
10    overflow_flag ,
11    scan_out0
12 );
13
14 input
15     reset ,           // system reset
16     clk ;             // system clock
17
18 input
19     scan_in0 ,        // test scan mode data input
20     scan_en ,         // test scan mode enable
21     test_mode ;       // test mode select
```

```
22
23 output
24     scan_out0;                // test scan mode data output
25
26 //inputs and outputs
27 input
28     [31:0] dividend, divisor;
29
30 output
31     [31:0] answer;
32
33 output reg
34     overflow_flag;
35
36 reg
37     stall_mc ,
38     stall_mc0 ,
39     stall_mc1 ,
40     stall_mc1_1 ,
41     stall_mc2 ,
42     stall_mc3 ,
43     stall_mc4;
44
45 reg
46     [11:0] yh_1 ,
47           yh_0;
48
49 reg
50     [10:0] yl_1 ,
51           yl_0;
52
53 reg
54     [11:0] address;
55
56 wire
57     [29:0] lut_output;
58
59 wire
60     [31:0] dividend_3 ,
61           divisor_3;
62 reg
```

```

63     [31:0] final_mult_1 ,
64         final_mult_2 ,
65         subtractor_input_1 ,
66         subtractor_input_2 ,
67         Ax_input_1 ,
68         Ax_input_2 ,
69         Ay_input_1 ,
70         Ax1_input_2 ,
71         Ay1_input_2 ,
72         Ay_input_2 ,
73         delay_in ,
74         dividend_0 ,
75         divisor_0 ,
76         dividend_1 ,
77         divisor_1 ,
78         dividend_2 ,
79         divisor_2 ,
80         dividend_4 ,
81         divisor_4 ,
82         a_input_1 ,
83         a_input_2 ;
84
85 wire
86     [31:0] final_output ,
87         subtractor_out_1 ,
88         Ax_output ,
89         Ay_output ,
90         a_output ,
91         delay_out ,
92         sub_module_output ;
93
94 mvk_stage_1 stage11(reset , clk , scan_in0 , scan_en , test_mode ,
95     address , lut_output , scan_out0);    //Calculate value from LUT
96
97 mvk_subtractor subtract11(reset , clk , scan_in0 , scan_en ,
98     test_mode , yh_1 , yl_1 , sub_module_output , scan_out0);
99
100 mvk_float_mult mult11(reset , clk , scan_in0 , scan_en , test_mode ,
101     a_input_1 , a_input_2 , a_output , scan_out0);    //Calculate A
102
103 mvk_float_mult mult12(reset , clk , scan_in0 , scan_en , test_mode ,
104     Ax_input_1 , Ax_input_2 , Ax_output , scan_out0);    //Calculate

```

```

    AX
100  mvk_float_mult mult13(reset , clk , scan_in0 , scan_en , test_mode ,
    Ay_input_1 , Ay_input_2 , Ay_output , scan_out0);    // Calculate
    AY
101
102  mvk_stage_2 stage21(reset , clk , scan_in0 , scan_en , test_mode ,
    delay_in , delay_out , scan_out0);
103
104  mvk_stage_2 stage22(reset , clk , scan_in0 , scan_en , test_mode ,
    dividend_2 , dividend_3 , scan_out0);
105
106  mvk_stage_2 stage23(reset , clk , scan_in0 , scan_en , test_mode ,
    divisor_2 , divisor_3 , scan_out0);
107
108  mvk_sub sub11(reset , clk , scan_in0 , scan_en , test_mode , scan_out0
    , subtractor_input_1 , subtractor_input_2 , subtractor_out_1);
109
110  mvk_float_mult mult14(reset , clk , scan_in0 , scan_en , test_mode ,
    final_mult_1 , final_mult_2 , final_output , scan_out0);    //
    Calculate AY
111
112  assign answer = (reset == 1'b1) ? 32'd0 : final_output;
113
114  always @(posedge clk or posedge reset)
115  begin
116      if(reset)
117      begin
118          overflow_flag <= 1'b1;
119          stall_mc <= 1'b1;
120          stall_mc0 <= 1'b0;
121          stall_mc1 <= 1'b0;
122          stall_mc1_1 <= 1'b0;
123          stall_mc2 <= 1'b0;
124          stall_mc3 <= 1'b0;
125          stall_mc4 <= 1'b0;
126      end
127      else
128      begin
129          if(stall_mc4 == 1'b1)    //We calculate (2-Ay)Ax which is
    the final Answer
130      begin

```

```

131         final_mult_1 <= subtractor_out_1;
132         final_mult_2 <= delay_out;
133     end
134
135     if(stall_mc3 == 1'b1)    //We calculate 2-Ay
136     begin
137         subtractor_input_1 <= 32'h4000_0000;
138         subtractor_input_2 <= Ay_output;
139         delay_in <= Ax_output;
140         stall_mc4 <= 1'b1;
141     end
142
143     if(stall_mc2 == 1'b1)    //We calculate Ax and Ay
144     begin
145         Ax_input_1 <= dividend_3;
146         Ax_input_2 <= a_output;
147         Ay_input_1 <= divisor_3;
148         Ay_input_2 <= a_output;
149         stall_mc3 <= 1'b1;
150     end
151
152     if(stall_mc1 == 1'b1)    //We calculate A
153     begin
154         a_input_1 <= {1'b0, lut_output};    //We call
155                                             Multiplier Module
156         a_input_2 <= sub_module_output;    //This is the
157                                             second input to mult module
158         dividend_2 <= dividend_1;
159         divisor_2 <= divisor_1;
160         stall_mc2 <= 1'b1;
161     end
162
163     if(stall_mc0 == 1'b1)    //In this stage we calculate (yh
164                             - yl) and also Get value from LUT
165     begin
166         yh_1 <= yh_0;    //Give input to yh - yl module
167         yl_1 <= yl_0;
168         stall_mc1 <= 1'b1;
169         dividend_1 <= dividend_0;
170         divisor_1 <= divisor_0;
171     end

```

```

169
170     if(stall_mc == 1'b1)    //In this stage we calculate (yh
    - yl) and also Get value from LUT
171     begin
172         dividend_0 <= dividend;
173         divisor_0 <= divisor;
174         yh_0 <= divisor[22:11];    //Give input to yh - yl
            module
175         yl_1 <= divisor[10:0];
176         address <= divisor[22:11];    //This the address we
            give to find 1/yh^2
177         stall_mc0 <= 1'b1;
178     end
179
180 end
181 end
182
183 endmodule // mvk_div

```

I.2 Floating Point Multiplier

```

1 module mvk_float_mult (
2     reset ,
3     clk ,
4     scan_in0 ,
5     scan_en ,
6     test_mode ,
7     input_a ,
8     input_b ,
9     output_z ,
10    scan_out0
11 );
12
13 input
14     reset ,    // system reset
15     clk ;    // system clock
16
17 input

```



```
18     scan_in0 ,           // test scan mode data input
19     scan_en ,           // test scan mode enable
20     test_mode ;         // test mode select
21
22 output
23     scan_out0 ;         // test scan mode data output
24
25 input
26     [31:0] input_a ;
27
28 input
29     [31:0] input_b ;
30
31 output
32     [31:0] output_z ;
33
34 reg
35     [3:0] state ;
36
37 reg
38     [8:0] xyz ;
39
40 parameter
41     get_a          = 4'd0 ,
42     get_b          = 4'd1 ,
43     unpack         = 4'd2 ,
44     check_nan_case = 4'd3 ,
45     normalise_a    = 4'd4 ,
46     normalise_b    = 4'd5 ,
47     multiply_0     = 4'd6 ,
48     multiply_1     = 4'd7 ,
49     normalise_1    = 4'd8 ,
50     normalise_2    = 4'd9 ,
51     round          = 4'd10 ,
52     pack           = 4'd11 ,
53     put_z          = 4'd12 ;
54
55 reg
56     [31:0] a , b , z ;
57
58 reg
```

```

59      [23:0] a_m_1, b_m_1, a_m_2, b_m_2, a_m_3, b_m_3;
60
61  reg
62      [23:0] z_m_1, z_m_2, z_m_3, z_m_4, z_m_5;
63
64  reg
65      [9:0] a_e_1, b_e_1, a_e_2, b_e_2, a_e_3, b_e_3;
66
67  reg
68      [9:0] z_e_1, z_e_2, z_e_3, z_e_4, z_e_5;
69
70  reg
71      a_s_1, b_s_1, a_s_2, b_s_2, a_s_3, b_s_3, z_s_1, z_s_2, z_s_3
72      , z_s_4, z_s_5;
73  reg
74      guard_2, round_bit_2, sticky_2, guard_3, round_bit_3,
75      sticky_3, guard_4, round_bit_4, sticky_4;
76
77  reg
78      [49:0] product;
79  wire
80      product_guard,
81      product_round,
82      product_sticky;
83
84  reg
85      [31:0] s_output_z;
86
87  reg
88      [23:0] multiplicand, multiplier;
89
90  reg
91      stall_mc,
92      stall_mc0,
93      stall_mc1,
94      stall_mc2,
95      stall_mc3,
96      stall_mc4,
97      stall_mc5,

```

```
98     stall_mc6 ,
99     stall_mc7 ,
100    stall_mc8 ,
101    stall_mc9 ;
102
103 reg
104     special ;
105
106 //reg
107 //     [49:0] product_1 ;
108
109 //mvk_mult mult11(reset , clk , scan_in0 , scan_en , test_mode ,
110     multiplicand , multiplier , product , product_guard ,
111     product_round , product_sticky , scan_out0) ;
112
113 //mvk_mult mult11(reset , clk , scan_in0 , scan_en , test_mode ,
114     multiplicand , multiplier , product , scan_out0) ;
115
116 assign output_z = s_output_z ;
117
118 always @(posedge clk)
119 begin
120     if (reset == 1)
121     begin
122         state <= unpack ;
123         xyz = 0 ;
124         stall_mc <= 1'b0 ;
125         stall_mc0 <= 1'b1 ;
126         stall_mc1 <= 1'b1 ;
127         stall_mc2 <= 1'b1 ;
128         stall_mc3 <= 1'b1 ;
129         stall_mc4 <= 1'b1 ;
130         stall_mc5 <= 1'b1 ;
131         stall_mc6 <= 1'b1 ;
132         stall_mc7 <= 1'b1 ;
133         stall_mc8 <= 1'b1 ;
134         stall_mc9 <= 1'b1 ;
135         a_m_1 <= 0 ;
136         b_m_1 <= 0 ;
137         a_m_2 <= 0 ;
```

```

136         b_m_2 <= 0;
137         a_m_3 <= 0;
138         b_m_3 <= 0;
139         a_e_1 <= 0;
140         b_e_1 <= 0;
141         a_e_2 <= 0;
142         b_e_2 <= 0;
143         a_e_3 <= 0;
144         b_e_3 <= 0;
145         a_s_1 <= 0;
146         b_s_1 <= 0;
147         a_s_2 <= 0;
148         b_s_2 <= 0;
149         a_s_3 <= 0;
150         b_s_3 <= 0;
151     end
152     else
153     begin
154         if (stall_mc8 == 1'b0)
155         begin
156             s_output_z[22 : 0] <= z_m_5[22:0];
157             s_output_z[30 : 23] <= z_e_5[7:0] + 127;
158             s_output_z[31] <= z_s_5;
159             if ($signed(z_e_5) == -126 && z_m_5[23] == 0)
160             begin
161                 s_output_z[30 : 23] <= 0;
162             end
163             //if overflow occurs, return inf
164             if ($signed(z_e_5) > 127)
165             begin
166                 s_output_z[22 : 0] <= 0;
167                 s_output_z[30 : 23] <= 255;
168                 s_output_z[31] <= z_s_5;
169             end
170             stall_mc9 <= 1'b0;
171         end //End stall_mc8 block
172
173         if (stall_mc7 == 1'b0)
174         begin
175             /*
176                 if (guard_4 && (round_bit_4 | sticky_4 | z_m_4[0]))

```

[illegible]

```
218         guard_4 <= z_m_3[1];
219         round_bit_4 <= z_m_3[0];
220         sticky_4 <= sticky_3 | z_m_3[1];
221     end
222 3:
223     begin
224         z_e_4 <= z_e_3 + 3;
225         z_m_4 <= z_m_3 >> 3;
226         guard_4 <= z_m_3[2];
227         round_bit_4 <= z_m_3[1];
228         sticky_4 <= sticky_3 | z_m_3[2];
229     end
230 4:
231     begin
232         z_e_4 <= z_e_3 + 4;
233         z_m_4 <= z_m_3 >> 4;
234         guard_4 <= z_m_3[3];
235         round_bit_4 <= z_m_3[2];
236         sticky_4 <= sticky_3 | z_m_3[3];
237     end
238 5:
239     begin
240         z_e_4 <= z_e_3 + 5;
241         z_m_4 <= z_m_3 >> 5;
242         guard_4 <= z_m_3[4];
243         round_bit_4 <= z_m_3[3];
244         sticky_4 <= sticky_3 | z_m_3[4];
245     end
246 6:
247     begin
248         z_e_4 <= z_e_3 + 6;
249         z_m_4 <= z_m_3 >> 6;
250         guard_4 <= z_m_3[5];
251         round_bit_4 <= z_m_3[4];
252         sticky_4 <= sticky_3 | z_m_3[5];
253     end
254 7:
255     begin
256         z_e_4 <= z_e_3 + 7;
257         z_m_4 <= z_m_3 >> 7;
258         guard_4 <= z_m_3[6];
```

```
259         round_bit_4 <= z_m_3[5];
260         sticky_4 <= sticky_3 | z_m_3[6];
261     end
262     8:
263     begin
264         z_e_4 <= z_e_3 + 8;
265         z_m_4 <= z_m_3 >> 8;
266         guard_4 <= z_m_3[7];
267         round_bit_4 <= z_m_3[6];
268         sticky_4 <= sticky_3 | z_m_3[7];
269     end
270     9:
271     begin
272         z_e_4 <= z_e_3 + 9;
273         z_m_4 <= z_m_3 >> 9;
274         guard_4 <= z_m_3[8];
275         round_bit_4 <= z_m_3[7];
276         sticky_4 <= sticky_3 | z_m_3[8];
277     end
278     10:
279     begin
280         z_e_4 <= z_e_3 + 10;
281         z_m_4 <= z_m_3 >> 10;
282         guard_4 <= z_m_3[9];
283         round_bit_4 <= z_m_3[8];
284         sticky_4 <= sticky_3 | z_m_3[9];
285     end
286     11:
287     begin
288         z_e_4 <= z_e_3 + 11;
289         z_m_4 <= z_m_3 >> 11;
290         guard_4 <= z_m_3[10];
291         round_bit_4 <= z_m_3[9];
292         sticky_4 <= sticky_3 | z_m_3[10];
293     end
294     12:
295     begin
296         z_e_4 <= z_e_3 + 12;
297         z_m_4 <= z_m_3 >> 12;
298         guard_4 <= z_m_3[11];
299         round_bit_4 <= z_m_3[10];
```

```
300         sticky_4 <= sticky_3 | z_m_3[11];
301     end
302     13:
303     begin
304         z_e_4 <= z_e_3 + 13;
305         z_m_4 <= z_m_3 >> 13;
306         guard_4 <= z_m_3[12];
307         round_bit_4 <= z_m_3[11];
308         sticky_4 <= sticky_3 | z_m_3[12];
309     end
310     14:
311     begin
312         z_e_4 <= z_e_3 + 14;
313         z_m_4 <= z_m_3 >> 14;
314         guard_4 <= z_m_3[13];
315         round_bit_4 <= z_m_3[12];
316         sticky_4 <= sticky_3 | z_m_3[13];
317     end
318     15:
319     begin
320         z_e_4 <= z_e_3 + 15;
321         z_m_4 <= z_m_3 >> 15;
322         guard_4 <= z_m_3[14];
323         round_bit_4 <= z_m_3[13];
324         sticky_4 <= sticky_3 | z_m_3[14];
325     end
326     16:
327     begin
328         z_e_4 <= z_e_3 + 16;
329         z_m_4 <= z_m_3 >> 16;
330         guard_4 <= z_m_3[15];
331         round_bit_4 <= z_m_3[14];
332         sticky_4 <= sticky_3 | z_m_3[15];
333     end
334     17:
335     begin
336         z_e_4 <= z_e_3 + 17;
337         z_m_4 <= z_m_3 >> 17;
338         guard_4 <= z_m_3[16];
339         round_bit_4 <= z_m_3[15];
340         sticky_4 <= sticky_3 | z_m_3[16];
```



```

341     end
342 18:
343     begin
344         z_e_4 <= z_e_3 + 18;
345         z_m_4 <= z_m_3 >> 18;
346         guard_4 <= z_m_3[17];
347         round_bit_4 <= z_m_3[16];
348         sticky_4 <= sticky_3 | z_m_3[17];
349     end
350 19:
351     begin
352         z_e_4 <= z_e_3 + 19;
353         z_m_4 <= z_m_3 >> 19;
354         guard_4 <= z_m_3[18];
355         round_bit_4 <= z_m_3[17];
356         sticky_4 <= sticky_3 | z_m_3[18];
357     end
358 20:
359     begin
360         z_e_4 <= z_e_3 + 20;
361         z_m_4 <= z_m_3 >> 20;
362         guard_4 <= z_m_3[19];
363         round_bit_4 <= z_m_3[18];
364         sticky_4 <= sticky_3 | z_m_3[19];
365     end
366 21:
367     begin
368         z_e_4 <= z_e_3 + 21;
369         z_m_4 <= z_m_3 >> 21;
370         guard_4 <= z_m_3[20];
371         round_bit_4 <= z_m_3[19];
372         sticky_4 <= sticky_3 | z_m_3[20];
373     end
374 22:
375     begin
376         z_e_4 <= z_e_3 + 22;
377         z_m_4 <= z_m_3 >> 22;
378         guard_4 <= z_m_3[21];
379         round_bit_4 <= z_m_3[20];
380         sticky_4 <= sticky_3 | z_m_3[21];
381     end

```

```

382             23:
383             begin
384                 z_e_4 <= z_e_3 + 23;
385                 z_m_4 <= z_m_3 >> 23;
386                 guard_4 <= z_m_3[22];
387                 round_bit_4 <= z_m_3[21];
388                 sticky_4 <= sticky_3 | z_m_3[22];
389             end
390         endcase
391     end
392     else
393     begin
394         z_e_4 <= z_e_3;
395         z_m_4 <= z_m_3;
396         guard_4 <= guard_3;
397         round_bit_4 <= round_bit_3;
398         sticky_4 <= sticky_3;
399     end
400     //      state <= round;
401     z_s_4 <= z_s_3;
402     stall_mc7 <= 1'b0;
403 end //End stall_mc6 block
404
405 if(stall_mc5 == 1'b0)
406 begin
407     casex(z_m_2)
408     24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
409     begin
410     //      state <= normalise_2;
411         z_m_3 <= z_m_2;
412         z_e_3 <= z_e_2;
413         stall_mc6 <= 1'b0;
414     end
415     24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
416     begin
417         z_e_3 <= z_e_2 - 1;
418         z_m_3 <= z_m_2 << 1;
419         z_m_3[0] <= guard_2;
420         guard_3 <= round_bit_2;
421         round_bit_3 <= 0;
422     end

```

```

423      24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
424      begin
425          z_e_3 <= z_e_2 - 2;
426          z_m_3 <= z_m_2 << 2;
427          z_m_3[0] <= round_bit_2;
428          guard_3 <= 0;
429          round_bit_3 <= 0;
430      end
431      24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
432      begin
433          z_e_3 <= z_e_2 - 3;
434          z_m_3 <= z_m_2 << 3;
435          z_m_3[0] <= 0;
436          guard_3 <= 0;
437          round_bit_3 <= 0;
438      end
439      24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
440      begin
441          z_e_3 <= z_e_2 - 4;
442          z_m_3 <= z_m_2 << 4;
443          z_m_3[0] <= 0;
444          guard_3 <= 0;
445          round_bit_3 <= 0;
446      end
447      24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
448      begin
449          z_e_3 <= z_e_2 - 5;
450          z_m_3 <= z_m_2 << 5;
451          z_m_3[0] <= 0;
452          guard_3 <= 0;
453          round_bit_3 <= 0;
454      end
455      24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
456      begin
457          z_e_3 <= z_e_2 - 6;
458          z_m_3 <= z_m_2 << 6;
459          z_m_3[0] <= 0;
460          guard_3 <= 0;
461          round_bit_3 <= 0;
462      end
463      24'b0000_0001_xxxx_xxxx_xxxx_xxxx:

```

```

464         begin
465             z_e_3 <= z_e_2 - 7;
466             z_m_3 <= z_m_2 << 7;
467             z_m_3[0] <= 0;
468             guard_3 <= 0;
469             round_bit_3 <= 0;
470         end
471 24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
472     begin
473         z_e_3 <= z_e_2 - 8;
474         z_m_3 <= z_m_2 << 8;
475         z_m_3[0] <= 0;
476         guard_3 <= 0;
477         round_bit_3 <= 0;
478     end
479 24'b0000_0000_01xx_xxxx_xxxx_xxxx:
480     begin
481         z_e_3 <= z_e_2 - 9;
482         z_m_3 <= z_m_2 << 9;
483         z_m_3[0] <= 0;
484         guard_3 <= 0;
485         round_bit_3 <= 0;
486     end
487 24'b0000_0000_001x_xxxx_xxxx_xxxx:
488     begin
489         z_e_3 <= z_e_2 - 10;
490         z_m_3 <= z_m_2 << 10;
491         z_m_3[0] <= 0;
492         guard_3 <= 0;
493         round_bit_3 <= 0;
494     end
495 24'b0000_0000_0001_xxxx_xxxx_xxxx:
496     begin
497         z_e_3 <= z_e_2 - 11;
498         z_m_3 <= z_m_2 << 11;
499         z_m_3[0] <= 0;
500         guard_3 <= 0;
501         round_bit_3 <= 0;
502     end
503 24'b0000_0000_0000_1xxx_xxxx_xxxx:
504     begin

```

```

505         z_e_3 <= z_e_2 - 12;
506         z_m_3 <= z_m_2 << 12;
507         z_m_3[0] <= 0;
508         guard_3 <= 0;
509         round_bit_3 <= 0;
510     end
511 24'b0000_0000_0000_01xx_xxxx_xxxx:
512 begin
513     z_e_3 <= z_e_2 - 13;
514     z_m_3 <= z_m_2 << 13;
515     z_m_3[0] <= 0;
516     guard_3 <= 0;
517     round_bit_3 <= 0;
518 end
519 24'b0000_0000_0000_001x_xxxx_xxxx:
520 begin
521     z_e_3 <= z_e_2 - 14;
522     z_m_3 <= z_m_2 << 14;
523     z_m_3[0] <= 0;
524     guard_3 <= 0;
525     round_bit_3 <= 0;
526 end
527 24'b0000_0000_0000_0001_xxxx_xxxx:
528 begin
529     z_e_3 <= z_e_2 - 15;
530     z_m_3 <= z_m_2 << 15;
531     z_m_3[0] <= 0;
532     guard_3 <= 0;
533     round_bit_3 <= 0;
534 end
535 24'b0000_0000_0000_0000_1xxx_xxxx:
536 begin
537     z_e_3 <= z_e_2 - 16;
538     z_m_3 <= z_m_2 << 16;
539     z_m_3[0] <= 0;
540     guard_3 <= 0;
541     round_bit_3 <= 0;
542 end
543 24'b0000_0000_0000_0000_01xx_xxxx:
544 begin
545     z_e_3 <= z_e_2 - 17;

```

```

546         z_m_3 <= z_m_2 << 17;
547         z_m_3[0] <= 0;
548         guard_3 <= 0;
549         round_bit_3 <= 0;
550     end
551 24'b0000_0000_0000_0000_001x_xxxx:
552 begin
553     z_e_3 <= z_e_2 - 18;
554     z_m_3 <= z_m_2 << 18;
555     z_m_3[0] <= 0;
556     guard_3 <= 0;
557     round_bit_3 <= 0;
558 end
559 24'b0000_0000_0000_0000_0001_xxxx:
560 begin
561     z_e_3 <= z_e_2 - 19;
562     z_m_3 <= z_m_2 << 19;
563     z_m_3[0] <= 0;
564     guard_3 <= 0;
565     round_bit_3 <= 0;
566 end
567 24'b0000_0000_0000_0000_0000_1xxx:
568 begin
569     z_e_3 <= z_e_2 - 20;
570     z_m_3 <= z_m_2 << 20;
571     z_m_3[0] <= 0;
572     guard_3 <= 0;
573     round_bit_3 <= 0;
574 end
575 24'b0000_0000_0000_0000_0000_01xx:
576 begin
577     z_e_3 <= z_e_2 - 21;
578     z_m_3 <= z_m_2 << 21;
579     z_m_3[0] <= 0;
580     guard_3 <= 0;
581     round_bit_3 <= 0;
582 end
583 24'b0000_0000_0000_0000_0000_001x:
584 begin
585     z_e_3 <= z_e_2 - 22;
586     z_m_3 <= z_m_2 << 22;

```

```

587         z_m_3[0] <= 0;
588         guard_3 <= 0;
589         round_bit_3 <= 0;
590     end
591     24'b0000_0000_0000_0000_0000_0001:
592     begin
593         z_e_3 <= z_e_2 - 23;
594         z_m_3 <= z_m_2 << 23;
595         z_m_3[0] <= 0;
596         guard_3 <= 0;
597         round_bit_3 <= 0;
598     end
599     endcase
600 //         state <= normalise_2;
601         z_s_3 <= z_s_2;
602         sticky_3 <= sticky_2;
603         stall_mc6 <= 1'b0;
604 end //End stall_mc5 block
605
606 if(stall_mc4 == 1'b0)
607 begin
608     z_m_2 <= product[49:26];
609     guard_2 <= product[25];
610     round_bit_2 <= product[24];
611     sticky_2 <= (product[23:0] != 0);
612     z_e_2 <= z_e_1;
613     z_s_2 <= z_s_1;
614 //         state <= normalise_1;
615     stall_mc5 <= 1'b0;
616 end //End stall_mc4 block
617
618 if(stall_mc3 == 1'b0)
619 begin
620     z_s_1 <= a_s_3 ^ b_s_3;
621     z_e_1 <= a_e_3 + b_e_3 + 1;
622     product <= a_m_3 * b_m_3 * 4;
623 //         multiplicand <= a_m_3;
624 //         multiplier <= b_m_3;
625 //         state <= normalise_1;
626     stall_mc4 <= 1'b0;
627 end //End stall_mc3 block

```

```

628
629     if (stall_mc2 == 1'b0)
630     begin
631         casex(a_m_2)
632
633             24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
634             begin
635                 //             state <= multiply_0;
636                 a_m_3 <= a_m_2;
637                 a_e_3 <= a_e_2;
638             end
639
640             24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
641             begin
642                 a_m_3 <= a_m_2 << 1;
643                 a_e_3 <= a_e_2 - 1;
644             end
645             24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
646             begin
647                 a_m_3 <= a_m_2 << 2;
648                 a_e_3 <= a_e_2 - 2;
649             end
650             24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
651             begin
652                 a_m_3 <= a_m_2 << 3;
653                 a_e_3 <= a_e_2 - 3;
654             end
655             24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
656             begin
657                 a_m_3 <= a_m_2 << 4;
658                 a_e_3 <= a_e_2 - 4;
659             end
660             24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
661             begin
662                 a_m_3 <= a_m_2 << 5;
663                 a_e_3 <= a_e_2 - 5;
664             end
665             24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
666             begin
667                 a_m_3 <= a_m_2 << 6;
668                 a_e_3 <= a_e_2 - 6;

```



```
669         end
670         24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
671         begin
672             a_m_3 <= a_m_2 << 7;
673             a_e_3 <= a_e_2 - 7;
674         end
675         24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
676         begin
677             a_m_3 <= a_m_2 << 7;
678             a_e_3 <= a_e_2 - 7;
679         end
680         24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
681         begin
682             a_m_3 <= a_m_2 << 8;
683             a_e_3 <= a_e_2 - 8;
684         end
685         24'b0000_0000_01xx_xxxx_xxxx_xxxx:
686         begin
687             a_m_3 <= a_m_2 << 9;
688             a_e_3 <= a_e_2 - 9;
689         end
690         24'b0000_0000_001x_xxxx_xxxx_xxxx:
691         begin
692             a_m_3 <= a_m_2 << 10;
693             a_e_3 <= a_e_2 - 10;
694         end
695         24'b0000_0000_0001_xxxx_xxxx_xxxx:
696         begin
697             a_m_3 <= a_m_2 << 11;
698             a_e_3 <= a_e_2 - 11;
699         end
700         24'b0000_0000_0000_1xxx_xxxx_xxxx:
701         begin
702             a_m_3 <= a_m_2 << 12;
703             a_e_3 <= a_e_2 - 12;
704         end
705         24'b0000_0000_0000_01xx_xxxx_xxxx:
706         begin
707             a_m_3 <= a_m_2 << 13;
708             a_e_3 <= a_e_2 - 13;
709         end
```

```
710      24'b0000_0000_0000_001x_xxxx_xxxx:
711      begin
712          a_m_3 <= a_m_2 << 14;
713          a_e_3 <= a_e_2 - 14;
714      end
715      24'b0000_0000_0000_0001_xxxx_xxxx:
716      begin
717          a_m_3 <= a_m_2 << 15;
718          a_e_3 <= a_e_2 - 15;
719      end
720      24'b0000_0000_0000_0000_1xxx_xxxx:
721      begin
722          a_m_3 <= a_m_2 << 16;
723          a_e_3 <= a_e_2 - 16;
724      end
725      24'b0000_0000_0000_0000_01xx_xxxx:
726      begin
727          a_m_3 <= a_m_2 << 17;
728          a_e_3 <= a_e_2 - 17;
729      end
730      24'b0000_0000_0000_0000_001x_xxxx:
731      begin
732          a_m_3 <= a_m_2 << 18;
733          a_e_3 <= a_e_2 - 18;
734      end
735      24'b0000_0000_0000_0000_0001_xxxx:
736      begin
737          a_m_3 <= a_m_2 << 19;
738          a_e_3 <= a_e_2 - 19;
739      end
740      24'b0000_0000_0000_0000_0000_1xxx:
741      begin
742          a_m_3 <= a_m_2 << 20;
743          a_e_3 <= a_e_2 - 20;
744      end
745      24'b0000_0000_0000_0000_0000_01xx:
746      begin
747          a_m_3 <= a_m_2 << 21;
748          a_e_3 <= a_e_2 - 21;
749      end
750      24'b0000_0000_0000_0000_0000_001x:
```

```

751         begin
752             a_m_3 <= a_m_2 << 22;
753             a_e_3 <= a_e_2 - 22;
754         end
755     24'b0000_0000_0000_0000_0000_0001:
756         begin
757             a_m_3 <= a_m_2 << 23;
758             a_e_3 <= a_e_2 - 23;
759         end
760     endcase
761
762 /*
763     if (b_m[23])
764     begin
765         state <= multiply_0;
766     end
767     else
768     begin
769         b_m <= b_m << 1;
770         b_e <= b_e - 1;
771     end
772 */
773     casex(b_m_2)
774
775     24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
776     begin
777         //         state <= multiply_0;
778         b_m_3 <= b_m_2;
779         b_e_3 <= b_e_2;
780     end
781
782     24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
783     begin
784         b_m_3 <= b_m_2 << 1;
785         a_e_3 <= a_e_2 - 1;
786     end
787     24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
788     begin
789         b_m_3 <= b_m_2 << 2;
790         a_e_3 <= a_e_2 - 2;
791     end

```

```
792      24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
793      begin
794          b_m_3 <= b_m_2 << 3;
795          a_e_3 <= a_e_2 - 3;
796      end
797      24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
798      begin
799          b_m_3 <= b_m_2 << 4;
800          a_e_3 <= a_e_2 - 4;
801      end
802      24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
803      begin
804          b_m_3 <= b_m_2 << 5;
805          a_e_3 <= a_e_2 - 5;
806      end
807      24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
808      begin
809          b_m_3 <= b_m_2 << 6;
810          a_e_3 <= a_e_2 - 6;
811      end
812      24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
813      begin
814          b_m_3 <= b_m_2 << 7;
815          a_e_3 <= a_e_2 - 7;
816      end
817      24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
818      begin
819          b_m_3 <= b_m_2 << 7;
820          a_e_3 <= a_e_2 - 7;
821      end
822      24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
823      begin
824          b_m_3 <= b_m_2 << 8;
825          a_e_3 <= a_e_2 - 8;
826      end
827      24'b0000_0000_01xx_xxxx_xxxx_xxxx:
828      begin
829          b_m_3 <= b_m_2 << 9;
830          a_e_3 <= a_e_2 - 9;
831      end
832      24'b0000_0000_001x_xxxx_xxxx_xxxx:
```

```
833      begin
834          b_m_3 <= b_m_2 << 10;
835          a_e_3 <= a_e_2 - 10;
836      end
837      24'b0000_0000_0001_XXXX_XXXX_XXXX:
838      begin
839          b_m_3 <= b_m_2 << 11;
840          a_e_3 <= a_e_2 - 11;
841      end
842      24'b0000_0000_0000_1xxx_XXXX_XXXX:
843      begin
844          b_m_3 <= b_m_2 << 12;
845          a_e_3 <= a_e_2 - 12;
846      end
847      24'b0000_0000_0000_01xx_XXXX_XXXX:
848      begin
849          b_m_3 <= b_m_2 << 13;
850          a_e_3 <= a_e_2 - 13;
851      end
852      24'b0000_0000_0000_001x_XXXX_XXXX:
853      begin
854          b_m_3 <= b_m_2 << 14;
855          a_e_3 <= a_e_2 - 14;
856      end
857      24'b0000_0000_0000_0001_XXXX_XXXX:
858      begin
859          b_m_3 <= b_m_2 << 15;
860          a_e_3 <= a_e_2 - 15;
861      end
862      24'b0000_0000_0000_0000_1xxx_XXXX:
863      begin
864          b_m_3 <= b_m_2 << 16;
865          a_e_3 <= a_e_2 - 16;
866      end
867      24'b0000_0000_0000_0000_01xx_XXXX:
868      begin
869          b_m_3 <= b_m_2 << 17;
870          a_e_3 <= a_e_2 - 17;
871      end
872      24'b0000_0000_0000_0000_001x_XXXX:
873      begin
```

```

874         b_m_3 <= b_m_2 << 18;
875         a_e_3 <= a_e_2 - 18;
876     end
877     24'b0000_0000_0000_0000_0001_xxxx:
878     begin
879         b_m_3 <= b_m_2 << 19;
880         a_e_3 <= a_e_2 - 19;
881     end
882     24'b0000_0000_0000_0000_0000_1xxx:
883     begin
884         b_m_3 <= b_m_2 << 20;
885         a_e_3 <= a_e_2 - 20;
886     end
887     24'b0000_0000_0000_0000_0000_01xx:
888     begin
889         b_m_3 <= b_m_2 << 21;
890         a_e_3 <= a_e_2 - 21;
891     end
892     24'b0000_0000_0000_0000_0000_001x:
893     begin
894         b_m_3 <= b_m_2 << 22;
895         a_e_3 <= a_e_2 - 22;
896     end
897     24'b0000_0000_0000_0000_0000_0001:
898     begin
899         b_m_3 <= b_m_2 << 23;
900         a_e_3 <= a_e_2 - 23;
901     end
902     endcase
903 //     state <= multiply_0;
904     a_s_3 <= a_s_2;
905     b_s_3 <= b_s_2;
906     stall_mc3 <= 1'b0;
907
908 end //End stall_mc2 block
909
910 if(stall_mc1 == 1'b0)
911 begin
912     if ((a_e_1 == 128 && a_m_1 != 0) || (b_e_1 == 128 &&
913         b_m_1 != 0))

```

```

914         z[31] <= 1;
915         z[30:23] <= 255;
916         z[22] <= 1;
917         z[21:0] <= 0;
918     //         state <= put_z;
919         stall_mc2 <= 1'b0;
920         special <= 1'b1;
921         //if a is inf return inf
922     end
923     else if (a_e_1 == 128)
924     begin
925         z[31] <= a_s_1 ^ b_s_1;
926         z[30:23] <= 255;
927         z[22:0] <= 0;
928     //         state <= put_z;
929         stall_mc2 <= 1'b0;
930         special <= 1'b1;
931         //if b is zero return NaN
932         if ($signed(b_e_1 == -127) && (b_m_1 == 0))
933         begin
934             z[31] <= 1;
935             z[30:23] <= 255;
936             z[22] <= 1;
937             z[21:0] <= 0;
938         //         state <= put_z;
939             special <= 1'b1;
940             stall_mc2 <= 1'b0;
941         end
942         //if b is inf return inf
943     end
944     else if (b_e_1 == 128)
945     begin
946         z[31] <= a_s_1 ^ b_s_1;
947         z[30:23] <= 255;
948         z[22:0] <= 0;
949     //         state <= put_z;
950             special <= 1'b1;
951             stall_mc2 <= 1'b0;
952         //if a is zero return zero
953     end
954     else if (($signed(a_e_1) == -127) && (a_m_1 == 0))

```

```

955         begin
956             z[31] <= a_s_1 ^ b_s_1;
957             z[30:23] <= 0;
958             z[22:0] <= 0;
959         //         state <= put_z;
960             special <= 1'b1;
961             stall_mc2 <= 1'b0;
962         //if b is zero return zero
963     end
964     else if (($signed(b_e_1) == -127) && (b_m_1 == 0))
965     begin
966         z[31] <= a_s_1 ^ b_s_1;
967         z[30:23] <= 0;
968         z[22:0] <= 0;
969         //         state <= put_z;
970             special <= 1'b1;
971             stall_mc2 <= 1'b0;
972     end
973     else
974     begin
975         //Denormalised Number
976         if ($signed(a_e_1) == -127)
977         begin
978             a_e_2 <= -126;
979             a_m_2 <= a_m_1;
980             a_s_2 <= a_s_1;
981         end
982         else
983         begin
984             a_m_2[23] <= 1;
985             a_m_2[22:0] <= a_m_1[22:0];
986             a_e_2 <= a_e_1;
987             a_s_2 <= a_s_1;
988         end
989         //Denormalised Number
990         if ($signed(b_e_1) == -127)
991         begin
992             b_e_2 <= -126;
993             b_m_2 <= b_m_1;
994             b_s_2 <= b_s_1;
995         end

```

```

996         else
997         begin
998             b_m_2[23] <= 1;
999             b_m_2[22:0] <= b_m_1[22:0];
1000             b_e_2 <= b_e_1;
1001             b_s_2 <= b_s_1;
1002         end
1003 //         state <= normalise_a;
1004         special <= 1'b0;
1005         stall_mc2 <= 1'b0;
1006     end
1007 end //End stall_mc1 block
1008
1009 if(stall_mc0 == 1'b0)
1010 begin
1011     a_m_1 <= input_a[22 : 0];
1012     b_m_1 <= input_b[22 : 0];
1013 //     a_e_1 <= input_a[30 : 23] - 127;
1014 //     b_e_1 <= input_b[30 : 23] - 127;
1015     a_s_1 <= input_a[31];
1016     b_s_1 <= input_b[31];
1017     stall_mc1 <= 1'b0;
1018 end //End stall_mc0 block
1019
1020 if(stall_mc == 1'b0)
1021 begin
1022     a_e_1 <= input_a[30 : 23] - 127;
1023     b_e_1 <= input_b[30 : 23] - 127;
1024     stall_mc0 <= 1'b0;
1025 end //End stall_mc0 block
1026
1027
1028     end //End Else block of reset
1029 end //End always block
1030
1031 endmodule

```

I.3 Wallace Tree Multiplier

```

1  module mvk_mult (
2      reset ,
3      clk ,
4      scan_in0 ,
5      scan_en ,
6      test_mode ,
7      multiplicand ,
8      multiplier ,
9      product ,
10     scan_out0
11 );
12
13 input
14     reset ,                // system reset
15     clk ;                  // system clock
16
17 input
18     scan_in0 ,             // test scan mode data input
19     scan_en ,              // test scan mode enable
20     test_mode ;            // test mode select
21
22 output
23     scan_out0 ;            // test scan mode data output
24
25 //inputs and outputs
26 input
27     [23:0] multiplicand , multiplier ;
28
29 output
30     [49:0] product ;
31
32 wire S11,C11, S12,C12, S13,C13, S114,C114, S115,C115, S116,C116,
      S117,C117, S118,C118, S119,C119, S1110,C1110_inst, S1111,
      C1111_inst, S1112,C1112_inst, S1113,C1113_inst, S1114,
      C1114_inst, S1115,C1115_inst, S1116,C1116_inst, S1117,
      C1117_inst, S1118,C1118_inst, S1119,C1119_inst, S1120,
      C1120_inst, S1121,C1121_inst, S1122,C1122_inst, S1123,
      C1123_inst, S1124,C1124_inst, S1125,C1125_inst, S1126,

```

```

    C1126_inst , S1127 , C1127_inst , S1128 , C1128_inst , S1129 ,
    C1129_inst , S1130 , C1130_inst , S1131 , C1131_inst , S1132 ,
    C1132_inst , S1133 , C1133_inst , S1134 , C1134_inst , S1135 ,
    C1135_inst , S1136 , C1136_inst , S1137 , C1137_inst , S1138 ,
    C1138_inst , S1139 , C1139_inst , S1140 , C1140_inst , S1141 ,
    C1141_inst , S1142 , C1142_inst , S1143 , C1143_inst ;    // Contains
    all the Sum's partial product
33
34 wire S124 , C124 , S125 , C125 , S126 , C126 , S127 , C127 , S128 , C128 , S129 ,
    C129 , S1210 , C1210_inst , S1211 , C1211_inst , S1212 , C1212_inst ,
    S1213 , C1213_inst , S1214 , C1214_inst , S1215 , C1215_inst , S1216 ,
    C1216_inst , S1217 , C1217_inst , S1218 , C1218_inst , S1219 ,
    C1219_inst , S1220 , C1220_inst , S1221 , C1221_inst , S1222 ,
    C1222_inst , S1223 , C1223_inst , S1224 , C1224_inst , S1225 ,
    C1225_inst , S1226 , C1226_inst , S1227 , C1227_inst , S1228 ,
    C1228_inst , S1229 , C1229_inst , S1230 , C1230_inst , S1231 ,
    C1231_inst , S1232 , C1232_inst , S1233 , C1233_inst , S1234 ,
    C1234_inst , S1235 , C1235_inst , S1236 , C1236_inst , S1237 ,
    C1237_inst , S1238 , C1238_inst , S1239 , C1239_inst , S1240 ,
    C1240_inst ;
35
36 wire S137 , C137 , S138 , C138 , S139 , C139 , S1310 , C1310_inst , S1311 ,
    C1311_inst , S1312 , C1312_inst , S1313 , C1313_inst , S1314 ,
    C1314_inst , S1315 , C1315_inst , S1316 , C1316_inst , S1317 ,
    C1317_inst , S1318 , C1318_inst , S1319 , C1319_inst , S1320 ,
    C1320_inst , S1321 , C1321_inst , S1322 , C1322_inst , S1323 ,
    C1323_inst , S1324 , C1324_inst , S1325 , C1325_inst , S1326 ,
    C1326_inst , S1327 , C1327_inst , S1328 , C1328_inst , S1329 ,
    C1329_inst , S1330 , C1330_inst , S1331 , C1331_inst , S1332 ,
    C1332_inst , S1333 , C1333_inst , S1334 , C1334_inst , S1335 ,
    C1335_inst , S1336 , C1336_inst , S1337 , C1337_inst ;
37
38 wire S1410 , C1410_inst , S1411 , C1411_inst , S1412 , C1412_inst , S1413 ,
    C1413_inst , S1414 , C1414_inst , S1415 , C1415_inst , S1416 ,
    C1416_inst , S1417 , C1417_inst , S1418 , C1418_inst , S1419 ,
    C1419_inst , S1420 , C1420_inst , S1421 , C1421_inst , S1422 ,
    C1422_inst , S1423 , C1423_inst , S1424 , C1424_inst , S1425 ,
    C1425_inst , S1426 , C1426_inst , S1427 , C1427_inst , S1428 ,
    C1428_inst , S1429 , C1429_inst , S1430 , C1430_inst , S1431 ,
    C1431_inst , S1432 , C1432_inst , S1433 , C1433_inst , S1434 ,
    C1434_inst ;

```

```

39
40 wire S1513,C1513_inst, S1514,C1514_inst, S1515,C1515_inst, S1516,
    C1516_inst, S1517,C1517_inst, S1518,C1518_inst, S1519,
    C1519_inst, S1520,C1520_inst, S1521,C1521_inst, S1522,
    C1522_inst, S1523,C1523_inst, S1524,C1524_inst, S1525,
    C1525_inst, S1526,C1526_inst, S1527,C1527_inst, S1528,
    C1528_inst, S1529,C1529_inst, S1530,C1530_inst, S1531,
    C1531_inst;
41
42 wire S1616,C1616_inst, S1617,C1617_inst, S1618,C1618_inst, S1619,
    C1619_inst, S1620,C1620_inst, S1621,C1621_inst, S1622,
    C1622_inst, S1623,C1623_inst, S1624,C1624_inst, S1625,
    C1625_inst, S1626,C1626_inst, S1627,C1627_inst, S1628,
    C1628_inst;
43
44 wire S1719,C1719_inst, S1720,C1720_inst, S1721,C1721_inst, S1722,
    C1722_inst, S1723,C1723_inst, S1724,C1724_inst, S1725,
    C1725_inst, S1822,C1822_inst;
45
46 wire S22,C22, S23,C23, S24,C24, S25,C25, S216,C216, S217,C217,
    S218,C218, S219,C219, S2110,C2110_inst, S2111,C2111_inst,
    S2112,C2112_inst, S2113,C2113_inst, S2114,C2114_inst, S2115,
    C2115_inst, S2116,C2116_inst, S2117,C2117_inst, S2118,
    C2118_inst, S2119,C2119_inst, S2120,C2120_inst, S2121,
    C2121_inst, S2122,C2122_inst, S2123,C2123_inst, S2124,
    C2124_inst, S2125,C2125_inst, S2126,C2126_inst, S2127,
    C2127_inst, S2128,C2128_inst, S2129,C2129_inst, S2130,
    C2130_inst, S2131,C2131_inst, S2132,C2132_inst, S2133,
    C2133_inst, S2134,C2134_inst, S2135,C2135_inst, S2136,
    C2136_inst, S2137,C2137_inst, S2138,C2138_inst, S2139,
    C2139_inst, S2140,C2140_inst, S2141,C2141_inst;
47
48 wire S226,C226, S227,C227, S228,C228, S229,C229, S2210,C2210_inst
    , S2211,C2211_inst, S2212,C2212_inst, S2213,C2213_inst, S2214,
    C2214_inst, S2215,C2215_inst, S2216,C2216_inst, S2217,
    C2217_inst, S2218,C2218_inst, S2219,C2219_inst, S2220,
    C2220_inst, S2221,C2221_inst, S2222,C2222_inst, S2223,
    C2223_inst, S2224,C2224_inst, S2225,C2225_inst, S2226,
    C2226_inst, S2227,C2227_inst, S2228,C2228_inst, S2229,
    C2229_inst, S2230,C2230_inst, S2231,C2231_inst, S2232,
    C2232_inst, S2233,C2233_inst, S2234,C2234_inst, S2235,

```

```

    C2235_inst , S2236 , C2236_inst , S2237 , C2237_inst , S2238 ,
    C2238_inst ;
49
50 wire S2311 , C2311_inst , S2312 , C2312_inst , S2313 , C2313_inst , S2314 ,
    C2314_inst , S2315 , C2315_inst , S2316 , C2316_inst , S2317 ,
    C2317_inst , S2318 , C2318_inst , S2319 , C2319_inst , S2320 ,
    C2320_inst , S2321 , C2321_inst , S2322 , C2322_inst , S2323 ,
    C2323_inst , S2324 , C2324_inst , S2325 , C2325_inst , S2326 ,
    C2326_inst , S2327 , C2327_inst , S2328 , C2328_inst , S2329 ,
    C2329_inst , S2330 , C2330_inst , S2331 , C2331_inst , S2332 ,
    C2332_inst , S2333 , C2333_inst , S2334 , C2334_inst , S2335 ,
    C2335_inst ;
51
52 wire S2415 , C2415_inst , S2416 , C2416_inst , S2417 , C2417_inst , S2418 ,
    C2418_inst , S2419 , C2419_inst , S2420 , C2420_inst , S2421 ,
    C2421_inst , S2422 , C2422_inst , S2423 , C2423_inst , S2424 ,
    C2424_inst , S2425 , C2425_inst , S2426 , C2426_inst , S2427 ,
    C2427_inst , S2428 , C2428_inst , S2429 , C2429_inst ;
53
54 wire S2520 , C2520_inst , S2521 , C2521_inst , S2522 , C2522_inst , S2523 ,
    C2523_inst , S2524 , C2524_inst , S2525 , C2525_inst , S2526 ,
    C2526_inst ;
55
56 wire S33 , C33 , S35 , C35 , S36 , C36 , S37 , C37 , S38 , C38 , S319 , C319 ,
    S3110 , C3110_inst , S3111 , C3111_inst , S3112 , C3112_inst , S3113 ,
    C3113_inst , S3114 , C3114_inst , S3115 , C3115_inst , S3116 ,
    C3116_inst , S3117 , C3117_inst , S3118 , C3118_inst , S3119 ,
    C3119_inst , S3120 , C3120_inst , S3121 , C3121_inst , S3122 ,
    C3122_inst , S3123 , C3123_inst , S3124 , C3124_inst , S3125 ,
    C3125_inst , S3126 , C3126_inst , S3127 , C3127_inst , S3128 ,
    C3128_inst , S3129 , C3129_inst , S3130 , C3130_inst , S3131 ,
    C3131_inst , S3132 , C3132_inst , S3133 , C3133_inst , S3134 ,
    C3134_inst , S3135 , C3135_inst , S3136 , C3136_inst , S3137 ,
    C3137_inst , S3138 , C3138_inst , S3139 , C3139_inst , S3140 ,
    C3140_inst , S3141 , C3141_inst , S3142 , C3142_inst , S3143 ,
    C3143_inst , S3144 , C3144_inst ;
57
58 wire S329 , C329 , S3210 , C3210_inst , S3211 , C3211_inst , S3212 ,
    C3212_inst , S3213 , C3213_inst , S3214 , C3214_inst , S3215 ,
    C3215_inst , S3216 , C3216_inst , S3217 , C3217_inst , S3218 ,
    C3218_inst , S3219 , C3219_inst , S3220 , C3220_inst , S3221 ,

```

```

    C3221_inst , S3222 , C3222_inst , S3223 , C3223_inst , S3224 ,
    C3224_inst , S3225 , C3225_inst , S3226 , C3226_inst , S3227 ,
    C3227_inst , S3228 , C3228_inst , S3229 , C3229_inst , S3230 ,
    C3230_inst , S3231 , C3231_inst , S3232 , C3232_inst , S3233 ,
    C3233_inst , S3234 , C3234_inst , S3235 , C3235_inst , S3236 ,
    C3236_inst ;
59
60 wire S3316 , C3316_inst , S3317 , C3317_inst , S3318 , C3318_inst , S3319 ,
    C3319_inst , S3320 , C3320_inst , S3321 , C3321_inst , S3322 ,
    C3322_inst , S3323 , C3323_inst , S3324 , C3324_inst , S3325 ,
    C3325_inst , S3326 , C3326_inst , S3327 , C3327_inst , S3328 ,
    C3328_inst , S3329 , C3329_inst , S3330 , C3330_inst , S3423 ,
    C3423_inst ;
61
62 wire S44 , C44 , S45 , C45 , S46 , C46 , S47 , C47 , S48 , C48 , S49 , C49 , S410 ,
    C410 , S411 , C411 , S412 , C412 , S413 , C413 , S4114 , C4114_inst , S4115 ,
    C4115_inst , S4116 , C4116_inst , S4117 , C4117_inst , S4118 ,
    C4118_inst , S4119 , C4119_inst , S4120 , C4120_inst , S4121 ,
    C4121_inst , S4122 , C4122_inst , S4123 , C4123_inst , S4124 ,
    C4124_inst , S4125 , C4125_inst , S4126 , C4126_inst , S4127 ,
    C4127_inst , S4128 , C4128_inst , S4129 , C4129_inst , S4130 ,
    C4130_inst , S4131 , C4131_inst , S4132 , C4132_inst , S4133 ,
    C4133_inst , S4134 , C4134_inst , S4135 , C4135_inst , S4136 ,
    C4136_inst , S4137 , C4137_inst , S4138 , C4138_inst , S4139 ,
    C4139_inst ;
63
64 wire S4214 , C4214_inst , S4215 , C4215_inst , S4216 , C4216_inst , S4217 ,
    C4217_inst , S4218 , C4218_inst , S4219 , C4219_inst , S4220 ,
    C4220_inst , S4221 , C4221_inst , S4222 , C4222_inst , S4223 ,
    C4223_inst , S4224 , C4224_inst , S4225 , C4225_inst , S4226 ,
    C4226_inst , S4227 , C4227_inst , S4228 , C4228_inst , S4229 ,
    C4229_inst , S4230 , C4230_inst , S4231 , C4231_inst , S4232 ,
    C4232_inst , S4324 , C4324_inst ;
65
66 wire S57 , C57 , S58 , C58 , S59 , C59 , S510 , C510 , S511 , C511 , S512 , C512 ,
    S513 , C513 , S514 , C514 , S515 , C515 , S516 , C516 , S517 , C517 , S518 ,
    C518 , S519 , C519 , S520 , C520 , S5121 , C5121_inst , S5122 , C5122_inst ,
    S5123 , C5123_inst , S5124 , C5124_inst , S5125 , C5125_inst , S5126 ,
    C5126_inst , S5127 , C5127_inst , S5128 , C5128_inst , S5129 ,
    C5129_inst , S5130 , C5130_inst , S5131 , C5131_inst , S5132 ,
    C5132_inst , S5133 , C5133_inst , S5134 , C5134_inst , S5135 ,

```

```

        C5135_inst , S5136 , C5136_inst , S5137 , C5137_inst ;
67
68 wire S5221 , C5221_inst , S5222 , C5222_inst , S5223 , C5223_inst , S5224 ,
        C5224_inst , S5225 , C5225_inst , S5226 , C5226_inst ;
69
70 wire S610 , C610 , S611 , C611 , S612 , C612 , S613 , C613 , S614 , C614 , S615 ,
        C615 , S616 , C616 , S617 , C617 , S618 , C618 , S619 , C619 , S620 , C620 ,
        S621 , C621 , S622 , C622 , S623 , C623 , S624 , C624 , S625 , C625 , S626 ,
        C626 , S627 , C627 , S628 , C628 , S629 , C629 , S630 , C630 , S631 , C631 ,
        S632 , C632 , S633 , C633 , S634 , C634 , S635 , C635 , S636 , C636 , S637 ,
        C637 , S638 , C638 , S639 , C639 , S640 , C640 , S641 , C641 , S642 , C642 ,
        S643 , C643 , S644 , C644 ;
71
72 wire S715 , C715 , S716 , C716 , S717 , C717 , S718 , C718 , S719 , C719 , S720 ,
        C720 , S721 , C721 , S722 , C722 , S723 , C723 , S724 , C724 , S725 , C725 ,
        S726 , C726 , S727 , C727 , S728 , C728 , S729 , C729 , S730 , C730 , S731 ,
        C731 , S732 , C732 , S733 , C733 , S734 , C734 , S735 , C735 , S736 , C736 ,
        S737 , C737 , S738 , C738 , S739 , C739 , S740 , C740 , S741 , C741 , S742 ,
        C742 , S743 , C743 , S744 , C744 , S745 , C745 ;
73
74 wire S822 , C822 , S823 , C823 , S824 , C824 , S825 , C825 , S826 , C826 , S827 ,
        C827 , S828 , C828 , S829 , C829 , S830 , C830 , S831 , C831 , S832 , C832 ,
        S833 , C833 , S834 , C834 , S835 , C835 , S836 , C836 , S837 , C837 , S838 ,
        C838 , S839 , C839 , S840 , C840 , S841 , C841 , S842 , C842 , S843 , C843 ,
        S844 , C844 , S845 , C845 ;
75
76 wire S923 , C923_inst , S924 , C924_inst , S925 , C925_inst , S926 ,
        C926_inst , S927 , C927_inst , S928 , C928_inst , S929 , C929_inst ,
        S930 , C930_inst , S931 , C931_inst , S932 , C932_inst , S933 , C933_inst
        , S934 , C934_inst , S935 , C935_inst , S936 , C936_inst , S937 ,
        C937_inst , S938 , C938_inst , S939 , C939_inst , S940 , C940_inst ,
        S941 , C941_inst , S942 , C942_inst , S943 , C943_inst , S944 , C944_inst
        , S945 , C945_inst , S946 , C946_inst ;
77
78 wire S1024 , C1024_inst , S1025 , C1025_inst , S1026 , C1026_inst , S1027 ,
        C1027_inst , S1028 , C1028_inst , S1029 , C1029_inst , S1030 ,
        C1030_inst , S1031 , C1031_inst , S1032 , C1032_inst , S1033 ,
        C1033_inst , S1034 , C1034_inst , S1035 , C1035_inst , S1036 ,
        C1036_inst , S1037 , C1037_inst , S1038 , C1038_inst , S1039 ,
        C1039_inst , S1040 , C1040_inst , S1041 , C1041_inst , S1042 ,
        C1042_inst , S1043 , C1043_inst , S1044 , C1044_inst , S1045 ,

```

```

    C1045_inst , S1046 , C1046_inst , S1047 , C1047_inst ;
79
80 wire [47:0] P0 , P1 , P2 , P3 , P4 , P5 , P6 , P7 , P8 , P9 , P10 , P11 , P12 ,
    P13 , P14 , P15 , P16 , P17 , P18 , P19 , P20 , P21 , P22 , P23 ;
81
82 wire sign ;
83 wire [7:0] exponent ,
84     exponent_1 ;
85
86 wire [7:0] multi_exponent ;
87 wire [7:0] multiplier_exponent ;
88
89 wire sum0 , sum1 , sum2 , sum3 , sum4 , sum5 , sum6 , sum7 ;
90 wire carry0 , carry1 , carry2 , carry3 , carry4 , carry5 , carry6 ,
    carry7 ;
91
92 wire
93     [49:0] product_2 ;
94
95 assign product = (reset == 1'b1) ? 48'b0 : {S1047 , S1046 , S1045 ,
    S1044 , S1043 , S1042 , S1041 , S1040 , S1039 , S1038 , S1037 , S1036 ,
    S1035 , S1034 , S1033 , S1032 , S1031 , S1030 , S1029 , S1028 , S1027
    , S1026 , S1025 , S1024 , S923 , S822 , S721 , S720 , S719 , S718 ,
    S717 , S716 , S715 , S614 , S613 , S612 , S611 , S610 , S59 , S58 , S57 ,
    S46 , S45 , S44 , S33 , S22 , S11 , P0[0]} ;
96
97 assign P0 = {24'd0 , multiplicand[23] & multiplier[0] ,
    multiplicand[22] & multiplier[0] , multiplicand[21] &
    multiplier[0] , multiplicand[20] & multiplier[0] , multiplicand
    [19] & multiplier[0] , multiplicand[18] & multiplier[0] ,
    multiplicand[17] & multiplier[0] , multiplicand[16] &
    multiplier[0] , multiplicand[15] & multiplier[0] , multiplicand
    [14] & multiplier[0] , multiplicand[13] & multiplier[0] ,
    multiplicand[12] & multiplier[0] , multiplicand[11] &
    multiplier[0] , multiplicand[10] & multiplier[0] , multiplicand
    [9] & multiplier[0] , multiplicand[8] & multiplier[0] ,
    multiplicand[7] & multiplier[0] , multiplicand[6] & multiplier
    [0] , multiplicand[5] & multiplier[0] , multiplicand[4] &
    multiplier[0] , multiplicand[3] & multiplier[0] , multiplicand
    [2] & multiplier[0] , multiplicand[1] & multiplier[0] ,
    multiplicand[0] & multiplier[0]} ;

```



```
98
99 assign P1 = {23'd0, multiplicand[23] & multiplier[1],
    multiplicand[22] & multiplier[1], multiplicand[21] &
    multiplier[1], multiplicand[20] & multiplier[1], multiplicand
    [19] & multiplier[1], multiplicand[18] & multiplier[1],
    multiplicand[17] & multiplier[1], multiplicand[16] &
    multiplier[1], multiplicand[15] & multiplier[1], multiplicand
    [14] & multiplier[1], multiplicand[13] & multiplier[1],
    multiplicand[12] & multiplier[1], multiplicand[11] &
    multiplier[1], multiplicand[10] & multiplier[1], multiplicand
    [9] & multiplier[1], multiplicand[8] & multiplier[1],
    multiplicand[7] & multiplier[1], multiplicand[6] & multiplier
    [1], multiplicand[5] & multiplier[1], multiplicand[4] &
    multiplier[1], multiplicand[3] & multiplier[1], multiplicand
    [2] & multiplier[1], multiplicand[1] & multiplier[1],
    multiplicand[0] & multiplier[1], 1'b0};
100
101 assign P2 = {22'd0, multiplicand[23] & multiplier[2],
    multiplicand[22] & multiplier[2], multiplicand[21] &
    multiplier[2], multiplicand[20] & multiplier[2], multiplicand
    [19] & multiplier[2], multiplicand[18] & multiplier[2],
    multiplicand[17] & multiplier[2], multiplicand[16] &
    multiplier[2], multiplicand[15] & multiplier[2], multiplicand
    [14] & multiplier[2], multiplicand[13] & multiplier[2],
    multiplicand[12] & multiplier[2], multiplicand[11] &
    multiplier[2], multiplicand[10] & multiplier[2], multiplicand
    [9] & multiplier[2], multiplicand[8] & multiplier[2],
    multiplicand[7] & multiplier[2], multiplicand[6] & multiplier
    [2], multiplicand[5] & multiplier[2], multiplicand[4] &
    multiplier[2], multiplicand[3] & multiplier[2], multiplicand
    [2] & multiplier[2], multiplicand[1] & multiplier[2],
    multiplicand[0] & multiplier[2], 1'b0, 1'b0};
102
103 assign P3 = {21'd0, multiplicand[23] & multiplier[3],
    multiplicand[22] & multiplier[3], multiplicand[21] &
    multiplier[3], multiplicand[20] & multiplier[3], multiplicand
    [19] & multiplier[3], multiplicand[18] & multiplier[3],
    multiplicand[17] & multiplier[3], multiplicand[16] &
    multiplier[3], multiplicand[15] & multiplier[3], multiplicand
    [14] & multiplier[3], multiplicand[13] & multiplier[3],
    multiplicand[12] & multiplier[3], multiplicand[11] &
```

```

multiplier[3], multiplicand[10] & multiplier[3], multiplicand
[9] & multiplier[3], multiplicand[8] & multiplier[3],
multiplicand[7] & multiplier[3], multiplicand[6] & multiplier
[3], multiplicand[5] & multiplier[3], multiplicand[4] &
multiplier[3], multiplicand[3] & multiplier[3], multiplicand
[2] & multiplier[3], multiplicand[1] & multiplier[3],
multiplicand[0] & multiplier[3], 3'd0};
104
105 assign P4 = {20'd0, multiplicand[23] & multiplier[4],
multiplicand[22] & multiplier[4], multiplicand[21] &
multiplier[4], multiplicand[20] & multiplier[4], multiplicand
[19] & multiplier[4], multiplicand[18] & multiplier[4],
multiplicand[17] & multiplier[4], multiplicand[16] &
multiplier[4], multiplicand[15] & multiplier[4], multiplicand
[14] & multiplier[4], multiplicand[13] & multiplier[4],
multiplicand[12] & multiplier[4], multiplicand[11] &
multiplier[4], multiplicand[10] & multiplier[4], multiplicand
[9] & multiplier[4], multiplicand[8] & multiplier[4],
multiplicand[7] & multiplier[4], multiplicand[6] & multiplier
[4], multiplicand[5] & multiplier[4], multiplicand[4] &
multiplier[4], multiplicand[3] & multiplier[4], multiplicand
[2] & multiplier[4], multiplicand[1] & multiplier[4],
multiplicand[0] & multiplier[4], 4'd0};
106
107 assign P5 = {19'd0, multiplicand[23] & multiplier[5],
multiplicand[22] & multiplier[5], multiplicand[21] &
multiplier[5], multiplicand[20] & multiplier[5], multiplicand
[19] & multiplier[5], multiplicand[18] & multiplier[5],
multiplicand[17] & multiplier[5], multiplicand[16] &
multiplier[5], multiplicand[15] & multiplier[5], multiplicand
[14] & multiplier[5], multiplicand[13] & multiplier[5],
multiplicand[12] & multiplier[5], multiplicand[11] &
multiplier[5], multiplicand[10] & multiplier[5], multiplicand
[9] & multiplier[5], multiplicand[8] & multiplier[5],
multiplicand[7] & multiplier[5], multiplicand[6] & multiplier
[5], multiplicand[5] & multiplier[5], multiplicand[4] &
multiplier[5], multiplicand[3] & multiplier[5], multiplicand
[2] & multiplier[5], multiplicand[1] & multiplier[5],
multiplicand[0] & multiplier[5], 5'd0};
108

```

```
109 assign P6 = {18'd0, multiplicand[23] & multiplier[6],
    multiplicand[22] & multiplier[6], multiplicand[21] &
    multiplier[6], multiplicand[20] & multiplier[6], multiplicand
    [19] & multiplier[6], multiplicand[18] & multiplier[6],
    multiplicand[17] & multiplier[6], multiplicand[16] &
    multiplier[6], multiplicand[15] & multiplier[6], multiplicand
    [14] & multiplier[6], multiplicand[13] & multiplier[6],
    multiplicand[12] & multiplier[6], multiplicand[11] &
    multiplier[6], multiplicand[10] & multiplier[6], multiplicand
    [9] & multiplier[6], multiplicand[8] & multiplier[6],
    multiplicand[7] & multiplier[6], multiplicand[6] & multiplier
    [6], multiplicand[5] & multiplier[6], multiplicand[4] &
    multiplier[6], multiplicand[3] & multiplier[6], multiplicand
    [2] & multiplier[6], multiplicand[1] & multiplier[6],
    multiplicand[0] & multiplier[6], 6'd0};
110
111 assign P7 = {17'd0, multiplicand[23] & multiplier[7],
    multiplicand[22] & multiplier[7], multiplicand[21] &
    multiplier[7], multiplicand[20] & multiplier[7], multiplicand
    [19] & multiplier[7], multiplicand[18] & multiplier[7],
    multiplicand[17] & multiplier[7], multiplicand[16] &
    multiplier[7], multiplicand[15] & multiplier[7], multiplicand
    [14] & multiplier[7], multiplicand[13] & multiplier[7],
    multiplicand[12] & multiplier[7], multiplicand[11] &
    multiplier[7], multiplicand[10] & multiplier[7], multiplicand
    [9] & multiplier[7], multiplicand[8] & multiplier[7],
    multiplicand[7] & multiplier[7], multiplicand[6] & multiplier
    [7], multiplicand[5] & multiplier[7], multiplicand[4] &
    multiplier[7], multiplicand[3] & multiplier[7], multiplicand
    [2] & multiplier[7], multiplicand[1] & multiplier[7],
    multiplicand[0] & multiplier[7], 7'd0};
112
113 assign P8 = {16'd0, multiplicand[23] & multiplier[8],
    multiplicand[22] & multiplier[8], multiplicand[21] &
    multiplier[8], multiplicand[20] & multiplier[8], multiplicand
    [19] & multiplier[8], multiplicand[18] & multiplier[8],
    multiplicand[17] & multiplier[8], multiplicand[16] &
    multiplier[8], multiplicand[15] & multiplier[8], multiplicand
    [14] & multiplier[8], multiplicand[13] & multiplier[8],
    multiplicand[12] & multiplier[8], multiplicand[11] &
    multiplier[8], multiplicand[10] & multiplier[8], multiplicand
```

```

[9] & multiplier[8], multiplicand[8] & multiplier[8],
multiplicand[7] & multiplier[8], multiplicand[6] & multiplier
[8], multiplicand[5] & multiplier[8], multiplicand[4] &
multiplier[8], multiplicand[3] & multiplier[8], multiplicand
[2] & multiplier[8], multiplicand[1] & multiplier[8],
multiplicand[0] & multiplier[8], 8'd0};
114
115 assign P9 = {15'd0, multiplicand[23] & multiplier[9],
multiplicand[22] & multiplier[9], multiplicand[21] &
multiplier[9], multiplicand[20] & multiplier[9], multiplicand
[19] & multiplier[9], multiplicand[18] & multiplier[9],
multiplicand[17] & multiplier[9], multiplicand[16] &
multiplier[9], multiplicand[15] & multiplier[9], multiplicand
[14] & multiplier[9], multiplicand[13] & multiplier[9],
multiplicand[12] & multiplier[9], multiplicand[11] &
multiplier[9], multiplicand[10] & multiplier[9], multiplicand
[9] & multiplier[9], multiplicand[8] & multiplier[9],
multiplicand[7] & multiplier[9], multiplicand[6] & multiplier
[9], multiplicand[5] & multiplier[9], multiplicand[4] &
multiplier[9], multiplicand[3] & multiplier[9], multiplicand
[2] & multiplier[9], multiplicand[1] & multiplier[9],
multiplicand[0] & multiplier[9], 9'd0};
116
117 assign P10 = {14'd0, multiplicand[23] & multiplier[10],
multiplicand[22] & multiplier[10], multiplicand[21] &
multiplier[10], multiplicand[20] & multiplier[10],
multiplicand[19] & multiplier[10], multiplicand[18] &
multiplier[10], multiplicand[17] & multiplier[10],
multiplicand[16] & multiplier[10], multiplicand[15] &
multiplier[10], multiplicand[14] & multiplier[10],
multiplicand[13] & multiplier[10], multiplicand[12] &
multiplier[10], multiplicand[11] & multiplier[10],
multiplicand[10] & multiplier[10], multiplicand[9] &
multiplier[10], multiplicand[8] & multiplier[10], multiplicand
[7] & multiplier[10], multiplicand[6] & multiplier[10],
multiplicand[5] & multiplier[10], multiplicand[4] & multiplier
[10], multiplicand[3] & multiplier[10], multiplicand[2] &
multiplier[10], multiplicand[1] & multiplier[10], multiplicand
[0] & multiplier[10], 10'd0};
118

```

```

119 assign P11 = {13'd0, multiplicand[23] & multiplier[11],
    multiplicand[22] & multiplier[11], multiplicand[21] &
    multiplier[11], multiplicand[20] & multiplier[11],
    multiplicand[19] & multiplier[11], multiplicand[18] &
    multiplier[11], multiplicand[17] & multiplier[11],
    multiplicand[16] & multiplier[11], multiplicand[15] &
    multiplier[11], multiplicand[14] & multiplier[11],
    multiplicand[13] & multiplier[11], multiplicand[12] &
    multiplier[11], multiplicand[11] & multiplier[11],
    multiplicand[10] & multiplier[11], multiplicand[9] &
    multiplier[11], multiplicand[8] & multiplier[11], multiplicand
    [7] & multiplier[11], multiplicand[6] & multiplier[11],
    multiplicand[5] & multiplier[11], multiplicand[4] & multiplier
    [11], multiplicand[3] & multiplier[11], multiplicand[2] &
    multiplier[11], multiplicand[1] & multiplier[11], multiplicand
    [0] & multiplier[11], 11'd0};
120
121 assign P12 = {12'd0, multiplicand[23] & multiplier[12],
    multiplicand[22] & multiplier[12], multiplicand[21] &
    multiplier[12], multiplicand[20] & multiplier[12],
    multiplicand[19] & multiplier[12], multiplicand[18] &
    multiplier[12], multiplicand[17] & multiplier[12],
    multiplicand[16] & multiplier[12], multiplicand[15] &
    multiplier[12], multiplicand[14] & multiplier[12],
    multiplicand[13] & multiplier[12], multiplicand[12] &
    multiplier[12], multiplicand[11] & multiplier[12],
    multiplicand[10] & multiplier[12], multiplicand[9] &
    multiplier[12], multiplicand[8] & multiplier[12], multiplicand
    [7] & multiplier[12], multiplicand[6] & multiplier[12],
    multiplicand[5] & multiplier[12], multiplicand[4] & multiplier
    [12], multiplicand[3] & multiplier[12], multiplicand[2] &
    multiplier[12], multiplicand[1] & multiplier[12], multiplicand
    [0] & multiplier[12], 12'd0};
122
123 assign P13 = {11'd0, multiplicand[23] & multiplier[13],
    multiplicand[22] & multiplier[13], multiplicand[21] &
    multiplier[13], multiplicand[20] & multiplier[13],
    multiplicand[19] & multiplier[13], multiplicand[18] &
    multiplier[13], multiplicand[17] & multiplier[13],
    multiplicand[16] & multiplier[13], multiplicand[15] &
    multiplier[13], multiplicand[14] & multiplier[13],

```

```

    multiplicand[13] & multiplier[13], multiplicand[12] &
    multiplier[13], multiplicand[11] & multiplier[13],
    multiplicand[10] & multiplier[13], multiplicand[9] &
    multiplier[13], multiplicand[8] & multiplier[13], multiplicand
    [7] & multiplier[13], multiplicand[6] & multiplier[13],
    multiplicand[5] & multiplier[13], multiplicand[4] & multiplier
    [13], multiplicand[3] & multiplier[13], multiplicand[2] &
    multiplier[13], multiplicand[1] & multiplier[13], multiplicand
    [0] & multiplier[13], 13'd0};
124
125 assign P14 = {10'd0, multiplicand[23] & multiplier[14],
    multiplicand[22] & multiplier[14], multiplicand[21] &
    multiplier[14], multiplicand[20] & multiplier[14],
    multiplicand[19] & multiplier[14], multiplicand[18] &
    multiplier[14], multiplicand[17] & multiplier[14],
    multiplicand[16] & multiplier[14], multiplicand[15] &
    multiplier[14], multiplicand[14] & multiplier[14],
    multiplicand[13] & multiplier[14], multiplicand[12] &
    multiplier[14], multiplicand[11] & multiplier[14],
    multiplicand[10] & multiplier[14], multiplicand[9] &
    multiplier[14], multiplicand[8] & multiplier[14], multiplicand
    [7] & multiplier[14], multiplicand[6] & multiplier[14],
    multiplicand[5] & multiplier[14], multiplicand[4] & multiplier
    [14], multiplicand[3] & multiplier[14], multiplicand[2] &
    multiplier[14], multiplicand[1] & multiplier[14], multiplicand
    [0] & multiplier[14], 14'd0};
126
127 assign P15 = {9'd0, multiplicand[23] & multiplier[15],
    multiplicand[22] & multiplier[15], multiplicand[21] &
    multiplier[15], multiplicand[20] & multiplier[15],
    multiplicand[19] & multiplier[15], multiplicand[18] &
    multiplier[15], multiplicand[17] & multiplier[15],
    multiplicand[16] & multiplier[15], multiplicand[15] &
    multiplier[15], multiplicand[14] & multiplier[15],
    multiplicand[13] & multiplier[15], multiplicand[12] &
    multiplier[15], multiplicand[11] & multiplier[15],
    multiplicand[10] & multiplier[15], multiplicand[9] &
    multiplier[15], multiplicand[8] & multiplier[15], multiplicand
    [7] & multiplier[15], multiplicand[6] & multiplier[15],
    multiplicand[5] & multiplier[15], multiplicand[4] & multiplier
    [15], multiplicand[3] & multiplier[15], multiplicand[2] &

```

```

    multiplier[15], multiplicand[1] & multiplier[15], multiplicand
    [0] & multiplier[15], 15'd0};
128
129 assign P16 = {8'd0, multiplicand[23] & multiplier[16],
    multiplicand[22] & multiplier[16], multiplicand[21] &
    multiplier[16], multiplicand[20] & multiplier[16],
    multiplicand[19] & multiplier[16], multiplicand[18] &
    multiplier[16], multiplicand[17] & multiplier[16],
    multiplicand[16] & multiplier[16], multiplicand[15] &
    multiplier[16], multiplicand[14] & multiplier[16],
    multiplicand[13] & multiplier[16], multiplicand[12] &
    multiplier[16], multiplicand[11] & multiplier[16],
    multiplicand[10] & multiplier[16], multiplicand[9] &
    multiplier[16], multiplicand[8] & multiplier[16], multiplicand
    [7] & multiplier[16], multiplicand[6] & multiplier[16],
    multiplicand[5] & multiplier[16], multiplicand[4] & multiplier
    [16], multiplicand[3] & multiplier[16], multiplicand[2] &
    multiplier[16], multiplicand[1] & multiplier[16], multiplicand
    [0] & multiplier[16], 16'd0};
130
131 assign P17 = {7'd0, multiplicand[23] & multiplier[17],
    multiplicand[22] & multiplier[17], multiplicand[21] &
    multiplier[17], multiplicand[20] & multiplier[17],
    multiplicand[19] & multiplier[17], multiplicand[18] &
    multiplier[17], multiplicand[17] & multiplier[17],
    multiplicand[16] & multiplier[17], multiplicand[15] &
    multiplier[17], multiplicand[14] & multiplier[17],
    multiplicand[13] & multiplier[17], multiplicand[12] &
    multiplier[17], multiplicand[11] & multiplier[17],
    multiplicand[10] & multiplier[17], multiplicand[9] &
    multiplier[17], multiplicand[8] & multiplier[17], multiplicand
    [7] & multiplier[17], multiplicand[6] & multiplier[17],
    multiplicand[5] & multiplier[17], multiplicand[4] & multiplier
    [17], multiplicand[3] & multiplier[17], multiplicand[2] &
    multiplier[17], multiplicand[1] & multiplier[17], multiplicand
    [0] & multiplier[17], 17'd0};
132
133 assign P18 = {6'd0, multiplicand[23] & multiplier[18],
    multiplicand[22] & multiplier[18], multiplicand[21] &
    multiplier[18], multiplicand[20] & multiplier[18],
    multiplicand[19] & multiplier[18], multiplicand[18] &

```

```

multiplier[18], multiplicand[17] & multiplier[18],
multiplicand[16] & multiplier[18], multiplicand[15] &
multiplier[18], multiplicand[14] & multiplier[18],
multiplicand[13] & multiplier[18], multiplicand[12] &
multiplier[18], multiplicand[11] & multiplier[18],
multiplicand[10] & multiplier[18], multiplicand[9] &
multiplier[18], multiplicand[8] & multiplier[18], multiplicand
[7] & multiplier[18], multiplicand[6] & multiplier[18],
multiplicand[5] & multiplier[18], multiplicand[4] & multiplier
[18], multiplicand[3] & multiplier[18], multiplicand[2] &
multiplier[18], multiplicand[1] & multiplier[18], multiplicand
[0] & multiplier[18], 18'd0};

```

134

```

135 assign P19 = {5'd0, multiplicand[23] & multiplier[19],
multiplicand[22] & multiplier[19], multiplicand[21] &
multiplier[19], multiplicand[20] & multiplier[19],
multiplicand[19] & multiplier[19], multiplicand[18] &
multiplier[19], multiplicand[17] & multiplier[19],
multiplicand[16] & multiplier[19], multiplicand[15] &
multiplier[19], multiplicand[14] & multiplier[19],
multiplicand[13] & multiplier[19], multiplicand[12] &
multiplier[19], multiplicand[11] & multiplier[19],
multiplicand[10] & multiplier[19], multiplicand[9] &
multiplier[19], multiplicand[8] & multiplier[19], multiplicand
[7] & multiplier[19], multiplicand[6] & multiplier[19],
multiplicand[5] & multiplier[19], multiplicand[4] & multiplier
[19], multiplicand[3] & multiplier[19], multiplicand[2] &
multiplier[19], multiplicand[1] & multiplier[19], multiplicand
[0] & multiplier[19], 19'd0};

```

136

```

137 assign P20 = {4'd0, multiplicand[23] & multiplier[20],
multiplicand[22] & multiplier[20], multiplicand[21] &
multiplier[20], multiplicand[20] & multiplier[20],
multiplicand[19] & multiplier[20], multiplicand[18] &
multiplier[20], multiplicand[17] & multiplier[20],
multiplicand[16] & multiplier[20], multiplicand[15] &
multiplier[20], multiplicand[14] & multiplier[20],
multiplicand[13] & multiplier[20], multiplicand[12] &
multiplier[20], multiplicand[11] & multiplier[20],
multiplicand[10] & multiplier[20], multiplicand[9] &
multiplier[20], multiplicand[8] & multiplier[20], multiplicand

```



```

[7] & multiplier[20], multiplicand[6] & multiplier[20],
multiplicand[5] & multiplier[20], multiplicand[4] & multiplier
[20], multiplicand[3] & multiplier[20], multiplicand[2] &
multiplier[20], multiplicand[1] & multiplier[20], multiplicand
[0] & multiplier[20], 20'd0};
138
139 assign P21 = {3'd0, multiplicand[23] & multiplier[21],
multiplicand[22] & multiplier[21], multiplicand[21] &
multiplier[21], multiplicand[20] & multiplier[21],
multiplicand[19] & multiplier[21], multiplicand[18] &
multiplier[21], multiplicand[17] & multiplier[21],
multiplicand[16] & multiplier[21], multiplicand[15] &
multiplier[21], multiplicand[14] & multiplier[21],
multiplicand[13] & multiplier[21], multiplicand[12] &
multiplier[21], multiplicand[11] & multiplier[21],
multiplicand[10] & multiplier[21], multiplicand[9] &
multiplier[21], multiplicand[8] & multiplier[21], multiplicand
[7] & multiplier[21], multiplicand[6] & multiplier[21],
multiplicand[5] & multiplier[21], multiplicand[4] & multiplier
[21], multiplicand[3] & multiplier[21], multiplicand[2] &
multiplier[21], multiplicand[1] & multiplier[21], multiplicand
[0] & multiplier[21], 21'd0};
140
141 assign P22 = {2'd0, multiplicand[23] & multiplier[22],
multiplicand[22] & multiplier[22], multiplicand[21] &
multiplier[22], multiplicand[20] & multiplier[22],
multiplicand[19] & multiplier[22], multiplicand[18] &
multiplier[22], multiplicand[17] & multiplier[22],
multiplicand[16] & multiplier[22], multiplicand[15] &
multiplier[22], multiplicand[14] & multiplier[22],
multiplicand[13] & multiplier[22], multiplicand[12] &
multiplier[22], multiplicand[11] & multiplier[22],
multiplicand[10] & multiplier[22], multiplicand[9] &
multiplier[22], multiplicand[8] & multiplier[22], multiplicand
[7] & multiplier[22], multiplicand[6] & multiplier[22],
multiplicand[5] & multiplier[22], multiplicand[4] & multiplier
[22], multiplicand[3] & multiplier[22], multiplicand[2] &
multiplier[22], multiplicand[1] & multiplier[22], multiplicand
[0] & multiplier[22], 22'd0};
142

```

```

143 assign P23 = {1'd0, multiplicand[23] & multiplier[23],
    multiplicand[22] & multiplier[23], multiplicand[21] &
    multiplier[23], multiplicand[20] & multiplier[23],
    multiplicand[19] & multiplier[23], multiplicand[18] &
    multiplier[23], multiplicand[17] & multiplier[23],
    multiplicand[16] & multiplier[23], multiplicand[15] &
    multiplier[23], multiplicand[14] & multiplier[23],
    multiplicand[13] & multiplier[23], multiplicand[12] &
    multiplier[23], multiplicand[11] & multiplier[23],
    multiplicand[10] & multiplier[23], multiplicand[9] &
    multiplier[23], multiplicand[8] & multiplier[23], multiplicand
    [7] & multiplier[23], multiplicand[6] & multiplier[23],
    multiplicand[5] & multiplier[23], multiplicand[4] & multiplier
    [23], multiplicand[3] & multiplier[23], multiplicand[2] &
    multiplier[23], multiplicand[1] & multiplier[23], multiplicand
    [0] & multiplier[23], 23'd0};

144
145 assign S11 = P0[1] ^ P1[1];
146 assign C11 = P0[1] & P1[1];
147 assign S12 = P0[2] ^ P1[2] ^ P2[2];
148 assign C12 = (P0[2] & P1[2]) | (P1[2] & P2[2]) | (P0[2] & P2[2]);
149 assign S13 = P0[3] ^ P1[3] ^ P2[3];
150 assign C13 = (P0[3] & P1[3]) | (P1[3] & P2[3]) | (P0[3] & P2[3]);
151 assign S114 = P0[4] ^ P1[4] ^ P2[4];
152 assign C114 = (P0[4] & P1[4]) | (P1[4] & P2[4]) | (P0[4] & P2[4])
    ;
153 assign S115 = P0[5] ^ P1[5] ^ P2[5];
154 assign C115 = (P0[5] & P1[5]) | (P1[5] & P2[5]) | (P0[5] & P2[5])
    ;
155 assign S116 = P0[6] ^ P1[6] ^ P2[6];
156 assign C116 = (P0[6] & P1[6]) | (P1[6] & P2[6]) | (P0[6] & P2[6])
    ;
157 assign S117 = P0[7] ^ P1[7] ^ P2[7];
158 assign C117 = (P0[7] & P1[7]) | (P1[7] & P2[7]) | (P0[7] & P2[7])
    ;
159 assign S118 = P0[8] ^ P1[8] ^ P2[8];
160 assign C118 = (P0[8] & P1[8]) | (P1[8] & P2[8]) | (P0[8] & P2[8])
    ;
161 assign S119 = P0[9] ^ P1[9] ^ P2[9];
162 assign C119 = (P0[9] & P1[9]) | (P1[9] & P2[9]) | (P0[9] & P2[9])
    ;

```

```

163 assign S1110 = P0[10] ^ P1[10] ^ P2[10];
164 assign C1110_inst = (P0[10] & P1[10]) | (P1[10] & P2[10]) | (P0
    [10] & P2[10]);
165 assign S1111 = P0[11] ^ P1[11] ^ P2[11];
166 assign C1111_inst = (P0[11] & P1[11]) | (P1[11] & P2[11]) | (P0
    [11] & P2[11]);
167 assign S1112 = P0[12] ^ P1[12] ^ P2[12];
168 assign C1112_inst = (P0[12] & P1[12]) | (P1[12] & P2[12]) | (P0
    [12] & P2[12]);
169 assign S1113 = P0[13] ^ P1[13] ^ P2[13];
170 assign C1113_inst = (P0[13] & P1[13]) | (P1[13] & P2[13]) | (P0
    [13] & P2[13]);
171 assign S1114 = P0[14] ^ P1[14] ^ P2[14];
172 assign C1114_inst = (P0[14] & P1[14]) | (P1[14] & P2[14]) | (P0
    [14] & P2[14]);
173 assign S1115 = P0[15] ^ P1[15] ^ P2[15];
174 assign C1115_inst = (P0[15] & P1[15]) | (P1[15] & P2[15]) | (P0
    [15] & P2[15]);
175 assign S1116 = P0[16] ^ P1[16] ^ P2[16];
176 assign C1116_inst = (P0[16] & P1[16]) | (P1[16] & P2[16]) | (P0
    [16] & P2[16]);
177 assign S1117 = P0[17] ^ P1[17] ^ P2[17];
178 assign C1117_inst = (P0[17] & P1[17]) | (P1[17] & P2[17]) | (P0
    [17] & P2[17]);
179 assign S1118 = P0[18] ^ P1[18] ^ P2[18];
180 assign C1118_inst = (P0[18] & P1[18]) | (P1[18] & P2[18]) | (P0
    [18] & P2[18]);
181 assign S1119 = P0[19] ^ P1[19] ^ P2[19];
182 assign C1119_inst = (P0[19] & P1[19]) | (P1[19] & P2[19]) | (P0
    [19] & P2[19]);
183 assign S1120 = P0[20] ^ P1[20] ^ P2[20];
184 assign C1120_inst = (P0[20] & P1[20]) | (P1[20] & P2[20]) | (P0
    [20] & P2[20]);
185 assign S1121 = P0[21] ^ P1[21] ^ P2[21];
186 assign C1121_inst = (P0[21] & P1[21]) | (P1[21] & P2[21]) | (P0
    [21] & P2[21]);
187 assign S1122 = P0[22] ^ P1[22] ^ P2[22];
188 assign C1122_inst = (P0[22] & P1[22]) | (P1[22] & P2[22]) | (P0
    [22] & P2[22]);
189 assign S1123 = P1[23] ^ P2[23] ^ P3[23];

```

```

190 assign C1123_inst = (P1[23] & P2[23]) | (P2[23] & P3[23]) | (P1
    [23] & P3[23]);
191 assign S1124 = P2[24] ^ P3[24] ^ P4[24];
192 assign C1124_inst = (P2[24] & P3[24]) | (P3[24] & P4[24]) | (P2
    [24] & P4[24]);
193 assign S1125 = P3[25] ^ P4[25] ^ P5[25];
194 assign C1125_inst = (P3[25] & P4[25]) | (P4[25] & P5[25]) | (P3
    [25] & P5[25]);
195 assign S1126 = P4[26] ^ P5[26] ^ P6[26];
196 assign C1126_inst = (P4[26] & P5[26]) | (P5[26] & P6[26]) | (P4
    [26] & P6[26]);
197 assign S1127 = P5[27] ^ P6[27] ^ P7[27];
198 assign C1127_inst = (P5[27] & P6[27]) | (P6[27] & P7[27]) | (P5
    [27] & P7[27]);
199 assign S1128 = P6[28] ^ P7[28] ^ P8[28];
200 assign C1128_inst = (P6[28] & P7[28]) | (P7[28] & P8[28]) | (P6
    [28] & P8[28]);
201 assign S1129 = P7[29] ^ P8[29] ^ P9[29];
202 assign C1129_inst = (P7[29] & P8[29]) | (P8[29] & P9[29]) | (P7
    [29] & P9[29]);
203 assign S1130 = P8[30] ^ P9[30] ^ P10[30];
204 assign C1130_inst = (P8[30] & P9[30]) | (P9[30] & P10[30]) | (P8
    [30] & P10[30]);
205 assign S1131 = P9[31] ^ P10[31] ^ P11[31];
206 assign C1131_inst = (P9[31] & P10[31]) | (P10[31] & P11[31]) | (
    P9[31] & P11[31]);
207 assign S1132 = P10[32] ^ P11[32] ^ P12[32];
208 assign C1132_inst = (P10[32] & P11[32]) | (P11[32] & P12[32]) | (
    P10[32] & P12[32]);
209 assign S1133 = P11[33] ^ P12[33] ^ P13[33];
210 assign C1133_inst = (P11[33] & P12[33]) | (P12[33] & P13[33]) | (
    P11[33] & P13[33]);
211 assign S1134 = P12[34] ^ P13[34] ^ P14[34];
212 assign C1134_inst = (P12[34] & P13[34]) | (P13[34] & P14[34]) | (
    P12[34] & P14[34]);
213 assign S1135 = P13[35] ^ P14[35] ^ P15[35];
214 assign C1135_inst = (P13[35] & P14[35]) | (P14[35] & P15[35]) | (
    P13[35] & P15[35]);
215 assign S1136 = P14[36] ^ P15[36] ^ P16[36];
216 assign C1136_inst = (P14[36] & P15[36]) | (P15[36] & P16[36]) | (
    P14[36] & P16[36]);

```

```

217 assign S1137 = P15[37] ^ P16[37] ^ P17[37];
218 assign C1137_inst = (P15[37] & P16[37]) | (P16[37] & P17[37]) | (
    P15[37] & P17[37]);
219 assign S1138 = P16[38] ^ P17[38] ^ P18[38];
220 assign C1138_inst = (P16[38] & P17[38]) | (P17[38] & P18[38]) | (
    P16[38] & P18[38]);
221 assign S1139 = P17[39] ^ P18[39] ^ P19[39];
222 assign C1139_inst = (P17[39] & P18[39]) | (P18[39] & P19[39]) | (
    P17[39] & P19[39]);
223 assign S1140 = P18[40] ^ P19[40] ^ P20[40];
224 assign C1140_inst = (P18[40] & P19[40]) | (P19[40] & P20[40]) | (
    P18[40] & P20[40]);
225 assign S1141 = P19[41] ^ P20[41] ^ P21[41];
226 assign C1141_inst = (P19[41] & P20[41]) | (P20[41] & P21[41]) | (
    P19[41] & P21[41]);
227 assign S1142 = P20[42] ^ P21[42] ^ P22[42];
228 assign C1142_inst = (P20[42] & P21[42]) | (P21[42] & P22[42]) | (
    P20[42] & P22[42]);
229 assign S1143 = P21[43] ^ P22[43];
230 assign C1143_inst = P21[43] & P22[43];
231
232 assign S124 = P3[4] ^ P4[4];
233 assign C124 = P3[4] & P4[4];
234 assign S125 = P3[5] ^ P4[5] ^ P5[5];
235 assign C125 = (P3[5] & P4[5]) | (P4[5] & P5[5]) | (P3[5] & P5[5])
    ;
236 assign S126 = P3[6] ^ P4[6] ^ P5[6];
237 assign C126 = (P3[6] & P4[6]) | (P4[6] & P5[6]) | (P3[6] & P5[6])
    ;
238 assign S127 = P3[7] ^ P4[7] ^ P5[7];
239 assign C127 = (P3[7] & P4[7]) | (P4[7] & P5[7]) | (P3[7] & P5[7])
    ;
240 assign S128 = P3[8] ^ P4[8] ^ P5[8];
241 assign C128 = (P3[8] & P4[8]) | (P4[8] & P5[8]) | (P3[8] & P5[8])
    ;
242 assign S129 = P3[9] ^ P4[9] ^ P5[9];
243 assign C129 = (P3[9] & P4[9]) | (P4[9] & P5[9]) | (P3[9] & P5[9])
    ;
244 assign S1210 = P3[10] ^ P4[10] ^ P5[10];
245 assign C1210_inst = (P3[10] & P4[10]) | (P4[10] & P5[10]) | (P3
    [10] & P5[10]);

```

```

246 assign S1211 = P3[11] ^ P4[11] ^ P5[11];
247 assign C1211_inst = (P3[11] & P4[11]) | (P4[11] & P5[11]) | (P3
    [11] & P5[11]);
248 assign S1212 = P3[12] ^ P4[12] ^ P5[12];
249 assign C1212_inst = (P3[12] & P4[12]) | (P4[12] & P5[12]) | (P3
    [12] & P5[12]);
250 assign S1213 = P3[13] ^ P4[13] ^ P5[13];
251 assign C1213_inst = (P3[13] & P4[13]) | (P4[13] & P5[13]) | (P3
    [13] & P5[13]);
252 assign S1214 = P3[14] ^ P4[14] ^ P5[14];
253 assign C1214_inst = (P3[14] & P4[14]) | (P4[14] & P5[14]) | (P3
    [14] & P5[14]);
254 assign S1215 = P3[15] ^ P4[15] ^ P5[15];
255 assign C1215_inst = (P3[15] & P4[15]) | (P4[15] & P5[15]) | (P3
    [15] & P5[15]);
256 assign S1216 = P3[16] ^ P4[16] ^ P5[16];
257 assign C1216_inst = (P3[16] & P4[16]) | (P4[16] & P5[16]) | (P3
    [16] & P5[16]);
258 assign S1217 = P3[17] ^ P4[17] ^ P5[17];
259 assign C1217_inst = (P3[17] & P4[17]) | (P4[17] & P5[17]) | (P3
    [17] & P5[17]);
260 assign S1218 = P3[18] ^ P4[18] ^ P5[18];
261 assign C1218_inst = (P3[18] & P4[18]) | (P4[18] & P5[18]) | (P3
    [18] & P5[18]);
262 assign S1219 = P3[19] ^ P4[19] ^ P5[19];
263 assign C1219_inst = (P3[19] & P4[19]) | (P4[19] & P5[19]) | (P3
    [19] & P5[19]);
264 assign S1220 = P3[20] ^ P4[20] ^ P5[20];
265 assign C1220_inst = (P3[20] & P4[20]) | (P4[20] & P5[20]) | (P3
    [20] & P5[20]);
266 assign S1221 = P3[21] ^ P4[21] ^ P5[21];
267 assign C1221_inst = (P3[21] & P4[21]) | (P4[21] & P5[21]) | (P3
    [21] & P5[21]);
268 assign S1222 = P3[22] ^ P4[22] ^ P5[22];
269 assign C1222_inst = (P3[22] & P4[22]) | (P4[22] & P5[22]) | (P3
    [22] & P5[22]);
270 assign S1223 = P4[23] ^ P5[23] ^ P6[23];
271 assign C1223_inst = (P4[23] & P5[23]) | (P5[23] & P6[23]) | (P4
    [23] & P6[23]);
272 assign S1224 = P5[24] ^ P6[24] ^ P7[24];

```

```

273 assign C1224_inst = (P5[24] & P6[24]) | (P6[24] & P7[24]) | (P5
    [24] & P7[24]);
274 assign S1225 = P6[25] ^ P7[25] ^ P8[25];
275 assign C1225_inst = (P6[25] & P7[25]) | (P7[25] & P8[25]) | (P6
    [25] & P8[25]);
276 assign S1226 = P7[26] ^ P8[26] ^ P9[26];
277 assign C1226_inst = (P7[26] & P8[26]) | (P8[26] & P9[26]) | (P7
    [26] & P9[26]);
278 assign S1227 = P8[27] ^ P9[27] ^ P10[27];
279 assign C1227_inst = (P8[27] & P9[27]) | (P9[27] & P10[27]) | (P8
    [27] & P10[27]);
280 assign S1228 = P9[28] ^ P10[28] ^ P11[28];
281 assign C1228_inst = (P9[28] & P10[28]) | (P10[28] & P11[28]) | (
    P9[28] & P11[28]);
282 assign S1229 = P10[29] ^ P11[29] ^ P12[29];
283 assign C1229_inst = (P10[29] & P11[29]) | (P11[29] & P12[29]) | (
    P10[29] & P12[29]);
284 assign S1230 = P11[30] ^ P12[30] ^ P13[30];
285 assign C1230_inst = (P11[30] & P12[30]) | (P12[30] & P13[30]) | (
    P11[30] & P13[30]);
286 assign S1231 = P12[31] ^ P13[31] ^ P14[31];
287 assign C1231_inst = (P12[31] & P13[31]) | (P13[31] & P14[31]) | (
    P12[31] & P14[31]);
288 assign S1232 = P13[32] ^ P14[32] ^ P15[32];
289 assign C1232_inst = (P13[32] & P14[32]) | (P14[32] & P15[32]) | (
    P13[32] & P15[32]);
290 assign S1233 = P14[33] ^ P15[33] ^ P16[33];
291 assign C1233_inst = (P14[33] & P15[33]) | (P15[33] & P16[33]) | (
    P14[33] & P16[33]);
292 assign S1234 = P15[34] ^ P16[34] ^ P17[34];
293 assign C1234_inst = (P15[34] & P16[34]) | (P16[34] & P17[34]) | (
    P15[34] & P17[34]);
294 assign S1235 = P16[35] ^ P17[35] ^ P18[35];
295 assign C1235_inst = (P16[35] & P17[35]) | (P17[35] & P18[35]) | (
    P16[35] & P18[35]);
296 assign S1236 = P17[36] ^ P18[36] ^ P19[36];
297 assign C1236_inst = (P17[36] & P18[36]) | (P18[36] & P19[36]) | (
    P17[36] & P19[36]);
298 assign S1237 = P18[37] ^ P19[37] ^ P20[37];
299 assign C1237_inst = (P18[37] & P19[37]) | (P19[37] & P20[37]) | (
    P18[37] & P20[37]);

```

```

300 assign S1238 = P19[38] ^ P20[38] ^ P21[38];
301 assign C1238_inst = (P19[38] & P20[38]) | (P20[38] & P21[38]) | (
    P19[38] & P21[38]);
302 assign S1239 = P20[39] ^ P21[39] ^ P22[39];
303 assign C1239_inst = (P20[39] & P21[39]) | (P21[39] & P22[39]) | (
    P20[39] & P22[39]);
304 assign S1240 = P21[40] ^ P22[40];
305 assign C1240_inst = P21[40] & P22[40];
306
307 assign S137 = P6[7] ^ P7[7];
308 assign C137 = P6[7] & P7[7];
309 assign S138 = P6[8] ^ P7[8] ^ P8[8];
310 assign C138 = (P6[8] & P7[8]) | (P7[8] & P8[8]) | (P6[8] & P8[8])
    ;
311 assign S139 = P6[9] ^ P7[9] ^ P8[9];
312 assign C139 = (P6[9] & P7[9]) | (P7[9] & P8[9]) | (P6[9] & P8[9])
    ;
313 assign S1310 = P6[10] ^ P7[10] ^ P8[10];
314 assign C1310_inst = (P6[10] & P7[10]) | (P7[10] & P8[10]) | (P6
    [10] & P8[10]);
315 assign S1311 = P6[11] ^ P7[11] ^ P8[11];
316 assign C1311_inst = (P6[11] & P7[11]) | (P7[11] & P8[11]) | (P6
    [11] & P8[11]);
317 assign S1312 = P6[12] ^ P7[12] ^ P8[12];
318 assign C1312_inst = (P6[12] & P7[12]) | (P7[12] & P8[12]) | (P6
    [12] & P8[12]);
319 assign S1313 = P6[13] ^ P7[13] ^ P8[13];
320 assign C1313_inst = (P6[13] & P7[13]) | (P7[13] & P8[13]) | (P6
    [13] & P8[13]);
321 assign S1314 = P6[14] ^ P7[14] ^ P8[14];
322 assign C1314_inst = (P6[14] & P7[14]) | (P7[14] & P8[14]) | (P6
    [14] & P8[14]);
323 assign S1315 = P6[15] ^ P7[15] ^ P8[15];
324 assign C1315_inst = (P6[15] & P7[15]) | (P7[15] & P8[15]) | (P6
    [15] & P8[15]);
325 assign S1316 = P6[16] ^ P7[16] ^ P8[16];
326 assign C1316_inst = (P6[16] & P7[16]) | (P7[16] & P8[16]) | (P6
    [16] & P8[16]);
327 assign S1317 = P6[17] ^ P7[17] ^ P8[17];
328 assign C1317_inst = (P6[17] & P7[17]) | (P7[17] & P8[17]) | (P6
    [17] & P8[17]);

```



```

329 assign S1318 = P6[18] ^ P7[18] ^ P8[18];
330 assign C1318_inst = (P6[18] & P7[18]) | (P7[18] & P8[18]) | (P6
    [18] & P8[18]);
331 assign S1319 = P6[19] ^ P7[19] ^ P8[19];
332 assign C1319_inst = (P6[19] & P7[19]) | (P7[19] & P8[19]) | (P6
    [19] & P8[19]);
333 assign S1320 = P6[20] ^ P7[20] ^ P8[20];
334 assign C1320_inst = (P6[20] & P7[20]) | (P7[20] & P8[20]) | (P6
    [20] & P8[20]);
335 assign S1321 = P6[21] ^ P7[21] ^ P8[21];
336 assign C1321_inst = (P6[21] & P7[21]) | (P7[21] & P8[21]) | (P6
    [21] & P8[21]);
337 assign S1322 = P6[22] ^ P7[22] ^ P8[22];
338 assign C1322_inst = (P6[22] & P7[22]) | (P7[22] & P8[22]) | (P6
    [22] & P8[22]);
339 assign S1323 = P7[23] ^ P8[23] ^ P9[23];
340 assign C1323_inst = (P7[23] & P8[23]) | (P8[23] & P9[23]) | (P7
    [23] & P9[23]);
341 assign S1324 = P8[24] ^ P9[24] ^ P10[24];
342 assign C1324_inst = (P8[24] & P9[24]) | (P9[24] & P10[24]) | (P8
    [24] & P10[24]);
343 assign S1325 = P9[25] ^ P10[25] ^ P11[25];
344 assign C1325_inst = (P9[25] & P10[25]) | (P10[25] & P11[25]) | (
    P9[25] & P11[25]);
345 assign S1326 = P10[26] ^ P11[26] ^ P12[26];
346 assign C1326_inst = (P10[26] & P11[26]) | (P11[26] & P12[26]) | (
    P10[26] & P12[26]);
347 assign S1327 = P11[27] ^ P12[27] ^ P13[27];
348 assign C1327_inst = (P11[27] & P12[27]) | (P12[27] & P13[27]) | (
    P11[27] & P13[27]);
349 assign S1328 = P12[28] ^ P13[28] ^ P14[28];
350 assign C1328_inst = (P12[28] & P13[28]) | (P13[28] & P14[28]) | (
    P12[28] & P14[28]);
351 assign S1329 = P13[29] ^ P14[29] ^ P15[29];
352 assign C1329_inst = (P13[29] & P14[29]) | (P14[29] & P15[29]) | (
    P13[29] & P15[29]);
353 assign S1330 = P14[30] ^ P15[30] ^ P16[30];
354 assign C1330_inst = (P14[30] & P15[30]) | (P15[30] & P16[30]) | (
    P14[30] & P16[30]);
355 assign S1331 = P15[31] ^ P16[31] ^ P17[31];

```

```

356 assign C1331_inst = (P15[31] & P16[31]) | (P16[31] & P17[31]) | (
    P15[31] & P17[31]);
357 assign S1332 = P16[32] ^ P17[32] ^ P18[32];
358 assign C1332_inst = (P16[32] & P17[32]) | (P17[32] & P18[32]) | (
    P16[32] & P18[32]);
359 assign S1333 = P17[33] ^ P18[33] ^ P19[33];
360 assign C1333_inst = (P17[33] & P18[33]) | (P18[33] & P19[33]) | (
    P17[33] & P19[33]);
361 assign S1334 = P18[34] ^ P19[34] ^ P20[34];
362 assign C1334_inst = (P18[34] & P19[34]) | (P19[34] & P20[34]) | (
    P18[34] & P20[34]);
363 assign S1335 = P19[35] ^ P20[35] ^ P21[35];
364 assign C1335_inst = (P19[35] & P20[35]) | (P20[35] & P21[35]) | (
    P19[35] & P21[35]);
365 assign S1336 = P20[36] ^ P21[36] ^ P22[36];
366 assign C1336_inst = (P20[36] & P21[36]) | (P21[36] & P22[36]) | (
    P20[36] & P22[36]);
367 assign S1337 = P21[37] ^ P22[37];
368 assign C1337_inst = P21[37] & P22[37];
369
370 assign S1410 = P9[10] ^ P10[10];
371 assign C1410_inst = P9[10] & P10[10];
372 assign S1411 = P9[11] ^ P10[11] ^ P11[11];
373 assign C1411_inst = (P9[11] & P10[11]) | (P10[11] & P11[11]) | (
    P9[11] & P11[11]);
374 assign S1412 = P9[12] ^ P10[12] ^ P11[12];
375 assign C1412_inst = (P9[12] & P10[12]) | (P10[12] & P11[12]) | (
    P9[12] & P11[12]);
376 assign S1413 = P9[13] ^ P10[13] ^ P11[13];
377 assign C1413_inst = (P9[13] & P10[13]) | (P10[13] & P11[13]) | (
    P9[13] & P11[13]);
378 assign S1414 = P9[14] ^ P10[14] ^ P11[14];
379 assign C1414_inst = (P9[14] & P10[14]) | (P10[14] & P11[14]) | (
    P9[14] & P11[14]);
380 assign S1415 = P9[15] ^ P10[15] ^ P11[15];
381 assign C1415_inst = (P9[15] & P10[15]) | (P10[15] & P11[15]) | (
    P9[15] & P11[15]);
382 assign S1416 = P9[16] ^ P10[16] ^ P11[16];
383 assign C1416_inst = (P9[16] & P10[16]) | (P10[16] & P11[16]) | (
    P9[16] & P11[16]);
384 assign S1417 = P9[17] ^ P10[17] ^ P11[17];

```

```

385 assign C1417_inst = (P9[17] & P10[17]) | (P10[17] & P11[17]) | (
    P9[17] & P11[17]);
386 assign S1418 = P9[18] ^ P10[18] ^ P11[18];
387 assign C1418_inst = (P9[18] & P10[18]) | (P10[18] & P11[18]) | (
    P9[18] & P11[18]);
388 assign S1419 = P9[19] ^ P10[19] ^ P11[19];
389 assign C1419_inst = (P9[19] & P10[19]) | (P10[19] & P11[19]) | (
    P9[19] & P11[19]);
390 assign S1420 = P9[20] ^ P10[20] ^ P11[20];
391 assign C1420_inst = (P9[20] & P10[20]) | (P10[20] & P11[20]) | (
    P9[20] & P11[20]);
392 assign S1421 = P9[21] ^ P10[21] ^ P11[21];
393 assign C1421_inst = (P9[21] & P10[21]) | (P10[21] & P11[21]) | (
    P9[21] & P11[21]);
394 assign S1422 = P9[22] ^ P10[22] ^ P11[22];
395 assign C1422_inst = (P9[22] & P10[22]) | (P10[22] & P11[22]) | (
    P9[22] & P11[22]);
396 assign S1423 = P10[23] ^ P11[23] ^ P12[23];
397 assign C1423_inst = (P10[23] & P11[23]) | (P11[23] & P12[23]) | (
    P10[23] & P12[23]);
398 assign S1424 = P11[24] ^ P12[24] ^ P13[24];
399 assign C1424_inst = (P11[24] & P12[24]) | (P12[24] & P13[24]) | (
    P11[24] & P13[24]);
400 assign S1425 = P12[25] ^ P13[25] ^ P14[25];
401 assign C1425_inst = (P12[25] & P13[25]) | (P13[25] & P14[25]) | (
    P12[25] & P14[25]);
402 assign S1426 = P13[26] ^ P14[26] ^ P15[26];
403 assign C1426_inst = (P13[26] & P14[26]) | (P14[26] & P15[26]) | (
    P13[26] & P15[26]);
404 assign S1427 = P14[27] ^ P15[27] ^ P16[27];
405 assign C1427_inst = (P14[27] & P15[27]) | (P15[27] & P16[27]) | (
    P14[27] & P16[27]);
406 assign S1428 = P15[28] ^ P16[28] ^ P17[28];
407 assign C1428_inst = (P15[28] & P16[28]) | (P16[28] & P17[28]) | (
    P15[28] & P17[28]);
408 assign S1429 = P16[29] ^ P17[29] ^ P18[29];
409 assign C1429_inst = (P16[29] & P17[29]) | (P17[29] & P18[29]) | (
    P16[29] & P18[29]);
410 assign S1430 = P17[30] ^ P18[30] ^ P19[30];
411 assign C1430_inst = (P17[30] & P18[30]) | (P18[30] & P19[30]) | (
    P17[30] & P19[30]);

```

```

412 assign S1431 = P18[31] ^ P19[31] ^ P20[31];
413 assign C1431_inst = (P18[31] & P19[31]) | (P19[31] & P20[31]) | (
    P18[31] & P20[31]);
414 assign S1432 = P19[32] ^ P20[32] ^ P21[32];
415 assign C1432_inst = (P19[32] & P20[32]) | (P20[32] & P21[32]) | (
    P19[32] & P21[32]);
416 assign S1433 = P20[33] ^ P21[33] ^ P22[33];
417 assign C1433_inst = (P20[33] & P21[33]) | (P21[33] & P22[33]) | (
    P20[33] & P22[33]);
418 assign S1434 = P21[34] ^ P22[34];
419 assign C1434_inst = P21[34] & P22[34];
420
421 assign S1513 = P12[13] ^ P13[13];
422 assign C1513_inst = P12[13] & P13[13];
423 assign S1514 = P12[14] ^ P13[14] ^ P14[14];
424 assign C1514_inst = (P12[14] & P13[14]) | (P13[14] & P14[14]) | (
    P12[14] & P14[14]);
425 assign S1515 = P12[15] ^ P13[15] ^ P14[15];
426 assign C1515_inst = (P12[15] & P13[15]) | (P13[15] & P14[15]) | (
    P12[15] & P14[15]);
427 assign S1516 = P12[16] ^ P13[16] ^ P14[16];
428 assign C1516_inst = (P12[16] & P13[16]) | (P13[16] & P14[16]) | (
    P12[16] & P14[16]);
429 assign S1517 = P12[17] ^ P13[17] ^ P14[17];
430 assign C1517_inst = (P12[17] & P13[17]) | (P13[17] & P14[17]) | (
    P12[17] & P14[17]);
431 assign S1518 = P12[18] ^ P13[18] ^ P14[18];
432 assign C1518_inst = (P12[18] & P13[18]) | (P13[18] & P14[18]) | (
    P12[18] & P14[18]);
433 assign S1519 = P12[19] ^ P13[19] ^ P14[19];
434 assign C1519_inst = (P12[19] & P13[19]) | (P13[19] & P14[19]) | (
    P12[19] & P14[19]);
435 assign S1520 = P12[20] ^ P13[20] ^ P14[20];
436 assign C1520_inst = (P12[20] & P13[20]) | (P13[20] & P14[20]) | (
    P12[20] & P14[20]);
437 assign S1521 = P12[21] ^ P13[21] ^ P14[21];
438 assign C1521_inst = (P12[21] & P13[21]) | (P13[21] & P14[21]) | (
    P12[21] & P14[21]);
439 assign S1522 = P12[22] ^ P13[22] ^ P14[22];
440 assign C1522_inst = (P12[22] & P13[22]) | (P13[22] & P14[22]) | (
    P12[22] & P14[22]);

```

```

441 assign S1523 = P13[23] ^ P14[23] ^ P15[23];
442 assign C1523_inst = (P13[23] & P14[23]) | (P14[23] & P15[23]) | (
    P13[23] & P15[23]);
443 assign S1524 = P14[24] ^ P15[24] ^ P16[24];
444 assign C1524_inst = (P14[24] & P15[24]) | (P15[24] & P16[24]) | (
    P14[24] & P16[24]);
445 assign S1525 = P15[25] ^ P16[25] ^ P17[25];
446 assign C1525_inst = (P15[25] & P16[25]) | (P16[25] & P17[25]) | (
    P15[25] & P17[25]);
447 assign S1526 = P16[26] ^ P17[26] ^ P18[26];
448 assign C1526_inst = (P16[26] & P17[26]) | (P17[26] & P18[26]) | (
    P16[26] & P18[26]);
449 assign S1527 = P17[27] ^ P18[27] ^ P19[27];
450 assign C1527_inst = (P17[27] & P18[27]) | (P18[27] & P19[27]) | (
    P17[27] & P19[27]);
451 assign S1528 = P18[28] ^ P19[28] ^ P20[28];
452 assign C1528_inst = (P18[28] & P19[28]) | (P19[28] & P20[28]) | (
    P18[28] & P20[28]);
453 assign S1529 = P19[29] ^ P20[29] ^ P21[29];
454 assign C1529_inst = (P19[29] & P20[29]) | (P20[29] & P21[29]) | (
    P19[29] & P21[29]);
455 assign S1530 = P20[30] ^ P21[30] ^ P22[30];
456 assign C1530_inst = (P20[30] & P21[30]) | (P21[30] & P22[30]) | (
    P20[30] & P22[30]);
457 assign S1531 = P21[31] ^ P22[31];
458 assign C1531_inst = P21[31] & P22[31];
459
460 assign S1616 = P15[16] ^ P16[16];
461 assign C1616_inst = P15[16] & P16[16];
462 assign S1617 = P15[17] ^ P16[17] ^ P17[17];
463 assign C1617_inst = (P15[17] & P16[17]) | (P16[17] & P17[17]) | (
    P15[17] & P17[17]);
464 assign S1618 = P15[18] ^ P16[18] ^ P17[18];
465 assign C1618_inst = (P15[18] & P16[18]) | (P16[18] & P17[18]) | (
    P15[18] & P17[18]);
466 assign S1619 = P15[19] ^ P16[19] ^ P17[19];
467 assign C1619_inst = (P15[19] & P16[19]) | (P16[19] & P17[19]) | (
    P15[19] & P17[19]);
468 assign S1620 = P15[20] ^ P16[20] ^ P17[20];
469 assign C1620_inst = (P15[20] & P16[20]) | (P16[20] & P17[20]) | (
    P15[20] & P17[20]);

```

```

470 assign S1621 = P15[21] ^ P16[21] ^ P17[21];
471 assign C1621_inst = (P15[21] & P16[21]) | (P16[21] & P17[21]) | (
    P15[21] & P17[21]);
472 assign S1622 = P15[22] ^ P16[22] ^ P17[22];
473 assign C1622_inst = (P15[22] & P16[22]) | (P16[22] & P17[22]) | (
    P15[22] & P17[22]);
474 assign S1623 = P16[23] ^ P17[23] ^ P18[23];
475 assign C1623_inst = (P16[23] & P17[23]) | (P17[23] & P18[23]) | (
    P16[23] & P18[23]);
476 assign S1624 = P17[24] ^ P18[24] ^ P19[24];
477 assign C1624_inst = (P17[24] & P18[24]) | (P18[24] & P19[24]) | (
    P17[24] & P19[24]);
478 assign S1625 = P18[25] ^ P19[25] ^ P20[25];
479 assign C1625_inst = (P18[25] & P19[25]) | (P19[25] & P20[25]) | (
    P18[25] & P20[25]);
480 assign S1626 = P19[26] ^ P20[26] ^ P21[26];
481 assign C1626_inst = (P19[26] & P20[26]) | (P20[26] & P21[26]) | (
    P19[26] & P21[26]);
482 assign S1627 = P20[27] ^ P21[27] ^ P22[27];
483 assign C1627_inst = (P20[27] & P21[27]) | (P21[27] & P22[27]) | (
    P20[27] & P22[27]);
484 assign S1628 = P21[28] ^ P22[28];
485 assign C1628_inst = P21[28] & P22[28];
486
487 assign S1719 = P18[19] ^ P19[19];
488 assign C1719_inst = P18[19] & P19[19];
489 assign S1720 = P18[20] ^ P19[20] ^ P20[20];
490 assign C1720_inst = (P18[20] & P19[20]) | (P19[20] & P20[20]) | (
    P18[20] & P20[20]);
491 assign S1721 = P18[21] ^ P19[21] ^ P20[21];
492 assign C1721_inst = (P18[21] & P19[21]) | (P19[21] & P20[21]) | (
    P18[21] & P20[21]);
493 assign S1722 = P18[22] ^ P19[22] ^ P20[22];
494 assign C1722_inst = (P18[22] & P19[22]) | (P19[22] & P20[22]) | (
    P18[22] & P20[22]);
495 assign S1723 = P19[23] ^ P20[23] ^ P21[23];
496 assign C1723_inst = (P19[23] & P20[23]) | (P20[23] & P21[23]) | (
    P19[23] & P21[23]);
497 assign S1724 = P20[24] ^ P21[24] ^ P22[24];
498 assign C1724_inst = (P20[24] & P21[24]) | (P21[24] & P22[24]) | (
    P20[24] & P22[24]);

```

```

499 assign S1725 = P21[25] ^ P22[25];
500 assign C1725_inst = P21[25] & P22[25];
501
502 assign S1822 = P21[22] ^ P22[22];
503 assign C1822_inst = P21[22] & P22[22];
504
505 //Second Stage
506
507 assign S22 = S12 ^ C11;
508 assign C22 = S12 & C11;
509 assign S23 = S13 ^ C12 ^ P3[3];
510 assign C23 = (S13 & C12) | (C12 & P3[3]) | (S13 & P3[3]);
511 assign S24 = S114 ^ C13 ^ S124;
512 assign C24 = (S114 & C13) | (C13 & S124) | (S114 & S124);
513 assign S25 = S115 ^ C114 ^ C124;
514 assign C25 = (S115 & C114) | (C114 & C124) | (S115 & C124);
515 assign S216 = S116 ^ C115 ^ C125;
516 assign C216 = (S116 & C115) | (C115 & C125) | (S116 & C125);
517 assign S217 = S117 ^ C116 ^ C126;
518 assign C217 = (S117 & C116) | (C116 & C126) | (S117 & C126);
519 assign S218 = S118 ^ C117 ^ C127;
520 assign C218 = (S118 & C117) | (C117 & C127) | (S118 & C127);
521 assign S219 = S119 ^ C118 ^ C128;
522 assign C219 = (S119 & C118) | (C118 & C128) | (S119 & C128);
523 assign S2110 = S1110 ^ C119 ^ C129;
524 assign C2110_inst = (S1110 & C119) | (C119 & C129) | (S1110 &
    C129);
525 assign S2111 = S1111 ^ C1110_inst ^ C1210_inst;
526 assign C2111_inst = (S1111 & C1110_inst) | (C1110_inst &
    C1210_inst) | (S1111 & C1210_inst);
527 assign S2112 = S1112 ^ C1111_inst ^ C1211_inst;
528 assign C2112_inst = (S1112 & C1111_inst) | (C1111_inst &
    C1211_inst) | (S1112 & C1211_inst);
529 assign S2113 = S1113 ^ C1112_inst ^ C1212_inst;
530 assign C2113_inst = (S1113 & C1112_inst) | (C1112_inst &
    C1212_inst) | (S1113 & C1212_inst);
531 assign S2114 = S1114 ^ C1113_inst ^ C1213_inst;
532 assign C2114_inst = (S1114 & C1113_inst) | (C1113_inst &
    C1213_inst) | (S1114 & C1213_inst);
533 assign S2115 = S1115 ^ C1114_inst ^ C1214_inst;

```

```

534 assign C2115_inst = (S1115 & C1114_inst) | (C1114_inst &
    C1214_inst) | (S1115 & C1214_inst);
535 assign S2116 = S1116 ^ C1115_inst ^ C1215_inst;
536 assign C2116_inst = (S1116 & C1115_inst) | (C1115_inst &
    C1215_inst) | (S1116 & C1215_inst);
537 assign S2117 = S1117 ^ C1116_inst ^ C1216_inst;
538 assign C2117_inst = (S1117 & C1116_inst) | (C1116_inst &
    C1216_inst) | (S1117 & C1216_inst);
539 assign S2118 = S1118 ^ C1117_inst ^ C1217_inst;
540 assign C2118_inst = (S1118 & C1117_inst) | (C1117_inst &
    C1217_inst) | (S1118 & C1217_inst);
541 assign S2119 = S1119 ^ C1118_inst ^ C1218_inst;
542 assign C2119_inst = (S1119 & C1118_inst) | (C1118_inst &
    C1218_inst) | (S1119 & C1218_inst);
543 assign S2120 = S1120 ^ C1119_inst ^ C1219_inst;
544 assign C2120_inst = (S1120 & C1119_inst) | (C1119_inst &
    C1219_inst) | (S1120 & C1219_inst);
545 assign S2121 = S1121 ^ C1120_inst ^ C1220_inst;
546 assign C2121_inst = (S1121 & C1120_inst) | (C1120_inst &
    C1220_inst) | (S1121 & C1220_inst);
547 assign S2122 = S1122 ^ C1121_inst ^ C1221_inst;
548 assign C2122_inst = (S1122 & C1121_inst) | (C1121_inst &
    C1221_inst) | (S1122 & C1221_inst);
549 assign S2123 = S1123 ^ C1122_inst ^ C1222_inst;
550 assign C2123_inst = (S1123 & C1122_inst) | (C1122_inst &
    C1222_inst) | (S1123 & C1222_inst);
551 assign S2124 = S1124 ^ C1123_inst ^ C1223_inst;
552 assign C2124_inst = (S1124 & C1123_inst) | (C1123_inst &
    C1223_inst) | (S1124 & C1223_inst);
553 assign S2125 = S1125 ^ C1124_inst ^ C1224_inst;
554 assign C2125_inst = (S1125 & C1124_inst) | (C1124_inst &
    C1224_inst) | (S1125 & C1224_inst);
555 assign S2126 = S1126 ^ C1125_inst ^ C1225_inst;
556 assign C2126_inst = (S1126 & C1125_inst) | (C1125_inst &
    C1225_inst) | (S1126 & C1225_inst);
557 assign S2127 = S1127 ^ C1126_inst ^ C1226_inst;
558 assign C2127_inst = (S1127 & C1126_inst) | (C1126_inst &
    C1226_inst) | (S1127 & C1226_inst);
559 assign S2128 = S1128 ^ C1127_inst ^ C1227_inst;
560 assign C2128_inst = (S1128 & C1127_inst) | (C1127_inst &
    C1227_inst) | (S1128 & C1227_inst);

```



```

561 assign S2129 = S1129 ^ C1128_inst ^ C1228_inst;
562 assign C2129_inst = (S1129 & C1128_inst) | (C1128_inst &
    C1228_inst) | (S1129 & C1228_inst);
563 assign S2130 = S1130 ^ C1129_inst ^ C1229_inst;
564 assign C2130_inst = (S1130 & C1129_inst) | (C1129_inst &
    C1229_inst) | (S1130 & C1229_inst);
565 assign S2131 = S1131 ^ C1130_inst ^ C1230_inst;
566 assign C2131_inst = (S1131 & C1130_inst) | (C1130_inst &
    C1230_inst) | (S1131 & C1230_inst);
567 assign S2132 = S1132 ^ C1131_inst ^ C1231_inst;
568 assign C2132_inst = (S1132 & C1131_inst) | (C1131_inst &
    C1231_inst) | (S1132 & C1231_inst);
569 assign S2133 = S1133 ^ C1132_inst ^ C1232_inst;
570 assign C2133_inst = (S1133 & C1132_inst) | (C1132_inst &
    C1232_inst) | (S1133 & C1232_inst);
571 assign S2134 = S1134 ^ C1133_inst ^ C1233_inst;
572 assign C2134_inst = (S1134 & C1133_inst) | (C1133_inst &
    C1233_inst) | (S1134 & C1233_inst);
573 assign S2135 = S1135 ^ C1134_inst ^ C1234_inst;
574 assign C2135_inst = (S1135 & C1134_inst) | (C1134_inst &
    C1234_inst) | (S1135 & C1234_inst);
575 assign S2136 = S1136 ^ C1135_inst ^ C1235_inst;
576 assign C2136_inst = (S1136 & C1135_inst) | (C1135_inst &
    C1235_inst) | (S1136 & C1235_inst);
577 assign S2137 = S1137 ^ C1136_inst ^ C1236_inst;
578 assign C2137_inst = (S1137 & C1136_inst) | (C1136_inst &
    C1236_inst) | (S1137 & C1236_inst);
579 assign S2138 = S1138 ^ C1137_inst ^ C1237_inst;
580 assign C2138_inst = (S1138 & C1137_inst) | (C1137_inst &
    C1237_inst) | (S1138 & C1237_inst);
581 assign S2139 = S1139 ^ C1138_inst ^ C1238_inst;
582 assign C2139_inst = (S1139 & C1138_inst) | (C1138_inst &
    C1238_inst) | (S1139 & C1238_inst);
583 assign S2140 = S1140 ^ C1139_inst ^ C1239_inst;
584 assign C2140_inst = (S1140 & C1139_inst) | (C1139_inst &
    C1239_inst) | (S1140 & C1239_inst);
585 assign S2141 = S1141 ^ C1140_inst ^ C1240_inst;
586 assign C2141_inst = (S1141 & C1140_inst) | (C1140_inst &
    C1240_inst) | (S1141 & C1240_inst);
587
588 assign S226 = S126 ^ P6[6];

```

```

589 assign C226 = S126 & P6[6];
590 assign S227 = S127 ^ S137;
591 assign C227 = S127 & S137;
592 assign S228 = C137 ^ S128 ^ S138;
593 assign C228 = (C137 & S128) | (S128 & S138) | (C137 & S138);
594 assign S229 = C138 ^ S129 ^ S139;
595 assign C229 = (C138 & S129) | (S129 & S139) | (C138 & S139);
596 assign S2210 = C139 ^ S1210 ^ S1310;
597 assign C2210_inst = (C139 & S1210) | (S1210 & S1310) | (C139 &
    S1310);
598 assign S2211 = C1310_inst ^ C1410_inst ^ S1211;
599 assign C2211_inst = (C1310_inst & C1410_inst) | (C1410_inst &
    S1211) | (C1310_inst & S1211);
600 assign S2212 = C1311_inst ^ C1411_inst ^ S1212;
601 assign C2212_inst = (C1311_inst & C1411_inst) | (C1411_inst &
    S1212) | (C1311_inst & S1212);
602 assign S2213 = C1312_inst ^ C1412_inst ^ S1213;
603 assign C2213_inst = (C1312_inst & C1412_inst) | (C1412_inst &
    S1213) | (C1312_inst & S1213);
604 assign S2214 = C1313_inst ^ C1413_inst ^ C1513_inst;
605 assign C2214_inst = (C1313_inst & C1413_inst) | (C1413_inst &
    C1513_inst) | (C1313_inst & C1513_inst);
606 assign S2215 = C1314_inst ^ C1414_inst ^ C1514_inst;
607 assign C2215_inst = (C1314_inst & C1414_inst) | (C1414_inst &
    C1514_inst) | (C1314_inst & C1514_inst);
608 assign S2216 = C1315_inst ^ C1415_inst ^ C1515_inst;
609 assign C2216_inst = (C1315_inst & C1415_inst) | (C1415_inst &
    C1515_inst) | (C1315_inst & C1515_inst);
610 assign S2217 = C1316_inst ^ C1416_inst ^ C1516_inst;
611 assign C2217_inst = (C1316_inst & C1416_inst) | (C1416_inst &
    C1516_inst) | (C1316_inst & C1516_inst);
612 assign S2218 = C1317_inst ^ C1417_inst ^ C1517_inst;
613 assign C2218_inst = (C1317_inst & C1417_inst) | (C1417_inst &
    C1517_inst) | (C1317_inst & C1517_inst);
614 assign S2219 = C1318_inst ^ C1418_inst ^ C1518_inst;
615 assign C2219_inst = (C1318_inst & C1418_inst) | (C1418_inst &
    C1518_inst) | (C1318_inst & C1518_inst);
616 assign S2220 = C1319_inst ^ C1419_inst ^ C1519_inst;
617 assign C2220_inst = (C1319_inst & C1419_inst) | (C1419_inst &
    C1519_inst) | (C1319_inst & C1519_inst);
618 assign S2221 = C1320_inst ^ C1420_inst ^ C1520_inst;

```

```

619 assign C2221_inst = (C1320_inst & C1420_inst) | (C1420_inst &
    C1520_inst) | (C1320_inst & C1520_inst);
620 assign S2222 = C1321_inst ^ C1421_inst ^ C1521_inst;
621 assign C2222_inst = (C1321_inst & C1421_inst) | (C1421_inst &
    C1521_inst) | (C1321_inst & C1521_inst);
622 assign S2223 = C1322_inst ^ C1422_inst ^ C1522_inst;
623 assign C2223_inst = (C1322_inst & C1422_inst) | (C1422_inst &
    C1522_inst) | (C1322_inst & C1522_inst);
624 assign S2224 = C1323_inst ^ C1423_inst ^ C1523_inst;
625 assign C2224_inst = (C1323_inst & C1423_inst) | (C1423_inst &
    C1523_inst) | (C1323_inst & C1523_inst);
626 assign S2225 = C1324_inst ^ C1424_inst ^ C1524_inst;
627 assign C2225_inst = (C1324_inst & C1424_inst) | (C1424_inst &
    C1524_inst) | (C1324_inst & C1524_inst);
628 assign S2226 = C1325_inst ^ C1425_inst ^ C1525_inst;
629 assign C2226_inst = (C1325_inst & C1425_inst) | (C1425_inst &
    C1525_inst) | (C1325_inst & C1525_inst);
630 assign S2227 = C1326_inst ^ C1426_inst ^ C1526_inst;
631 assign C2227_inst = (C1326_inst & C1426_inst) | (C1426_inst &
    C1526_inst) | (C1326_inst & C1526_inst);
632 assign S2228 = C1327_inst ^ C1427_inst ^ C1527_inst;
633 assign C2228_inst = (C1327_inst & C1427_inst) | (C1427_inst &
    C1527_inst) | (C1327_inst & C1527_inst);
634 assign S2229 = C1328_inst ^ C1428_inst ^ C1528_inst;
635 assign C2229_inst = (C1328_inst & C1428_inst) | (C1428_inst &
    C1528_inst) | (C1328_inst & C1528_inst);
636 assign S2230 = C1329_inst ^ C1429_inst ^ C1529_inst;
637 assign C2230_inst = (C1329_inst & C1429_inst) | (C1429_inst &
    C1529_inst) | (C1329_inst & C1529_inst);
638 assign S2231 = C1330_inst ^ C1430_inst ^ C1530_inst;
639 assign C2231_inst = (C1330_inst & C1430_inst) | (C1430_inst &
    C1530_inst) | (C1330_inst & C1530_inst);
640 assign S2232 = C1331_inst ^ C1431_inst ^ C1531_inst;
641 assign C2232_inst = (C1331_inst & C1431_inst) | (C1431_inst &
    C1531_inst) | (C1331_inst & C1531_inst);
642 assign S2233 = C1332_inst ^ C1432_inst ^ S1233;
643 assign C2233_inst = (C1332_inst & C1432_inst) | (C1432_inst &
    S1233) | (C1332_inst & S1233);
644 assign S2234 = C1333_inst ^ C1433_inst ^ S1234;
645 assign C2234_inst = (C1333_inst & C1433_inst) | (C1433_inst &
    S1234) | (C1333_inst & S1234);

```

```

646 assign S2235 = C1334_inst ^ C1434_inst ^ S1235;
647 assign C2235_inst = (C1334_inst & C1434_inst) | (C1434_inst &
    S1235) | (C1334_inst & S1235);
648 assign S2236 = C1335_inst ^ S1236 ^ S1336;
649 assign C2236_inst = (C1335_inst & S1236) | (S1236 & S1336) | (
    C1335_inst & S1336);
650 assign S2237 = C1336_inst ^ S1237 ^ S1337;
651 assign C2237_inst = (C1336_inst & S1237) | (S1237 & S1337) | (
    C1336_inst & S1337);
652 assign S2238 = C1337_inst ^ S1238 ^ P22[38];
653 assign C2238_inst = (C1337_inst & S1238) | (S1238 & P22[38]) | (
    C1337_inst & P22[38]);
654
655 assign S2311 = S1311 ^ S1411;
656 assign C2311_inst = S1311 & S1411;
657 assign S2312 = S1312 ^ S1412 ^ P12[12];
658 assign C2312_inst = (S1312 & S1412) | (S1412 & P12[12]) | (S1312
    & P12[12]);
659 assign S2313 = S1313 ^ S1413 ^ S1513;
660 assign C2313_inst = (S1313 & S1413) | (S1413 & S1513) | (S1313 &
    S1513);
661 assign S2314 = S1214 ^ S1314 ^ S1414;
662 assign C2314_inst = (S1214 & S1314) | (S1314 & S1414) | (S1214 &
    S1414);
663 assign S2315 = S1215 ^ S1315 ^ S1415;
664 assign C2315_inst = (S1215 & S1315) | (S1315 & S1415) | (S1215 &
    S1415);
665 assign S2316 = S1216 ^ S1316 ^ S1416;
666 assign C2316_inst = (S1216 & S1316) | (S1316 & S1416) | (S1216 &
    S1416);
667 assign S2317 = C1616_inst ^ S1217 ^ S1317;
668 assign C2317_inst = (C1616_inst & S1217) | (S1217 & S1317) | (
    C1616_inst & S1317);
669 assign S2318 = C1617_inst ^ S1218 ^ S1318;
670 assign C2318_inst = (C1617_inst & S1218) | (S1218 & S1318) | (
    C1617_inst & S1318);
671 assign S2319 = C1618_inst ^ S1219 ^ S1319;
672 assign C2319_inst = (C1618_inst & S1219) | (S1219 & S1319) | (
    C1618_inst & S1319);
673 assign S2320 = C1619_inst ^ C1719_inst ^ S1220;

```

```

674 assign C2320_inst = (C1619_inst & C1719_inst) | (C1719_inst &
    S1220) | (C1619_inst & S1220);
675 assign S2321 = C1620_inst ^ C1720_inst ^ S1221;
676 assign C2321_inst = (C1620_inst & C1720_inst) | (C1720_inst &
    S1221) | (C1620_inst & S1221);
677 assign S2322 = C1621_inst ^ C1721_inst ^ S1222;
678 assign C2322_inst = (C1621_inst & C1721_inst) | (C1721_inst &
    S1222) | (C1621_inst & S1222);
679 assign S2323 = C1622_inst ^ C1722_inst ^ C1822_inst;
680 assign C2323_inst = (C1622_inst & C1722_inst) | (C1722_inst &
    C1822_inst) | (C1622_inst & C1822_inst);
681 assign S2324 = C1623_inst ^ C1723_inst ^ S1224;
682 assign C2324_inst = (C1623_inst & C1723_inst) | (C1723_inst &
    S1224) | (C1623_inst & S1224);
683 assign S2325 = C1624_inst ^ C1724_inst ^ S1225;
684 assign C2325_inst = (C1624_inst & C1724_inst) | (C1724_inst &
    S1225) | (C1624_inst & S1225);
685 assign S2326 = C1625_inst ^ C1725_inst ^ S1226;
686 assign C2326_inst = (C1625_inst & C1725_inst) | (C1725_inst &
    S1226) | (C1625_inst & S1226);
687 assign S2327 = C1626_inst ^ S1227 ^ S1327;
688 assign C2327_inst = (C1626_inst & S1227) | (S1227 & S1327) | (
    C1626_inst & S1327);
689 assign S2328 = C1627_inst ^ S1228 ^ S1328;
690 assign C2328_inst = (C1627_inst & S1228) | (S1228 & S1328) | (
    C1627_inst & S1328);
691 assign S2329 = C1628_inst ^ S1229 ^ S1329;
692 assign C2329_inst = (C1628_inst & S1229) | (S1229 & S1329) | (
    C1628_inst & S1329);
693 assign S2330 = S1230 ^ S1330 ^ S1430;
694 assign C2330_inst = (S1230 & S1330) | (S1330 & S1430) | (S1230 &
    S1430);
695 assign S2331 = S1231 ^ S1331 ^ S1431;
696 assign C2331_inst = (S1231 & S1331) | (S1331 & S1431) | (S1231 &
    S1431);
697 assign S2332 = S1232 ^ S1332 ^ S1432;
698 assign C2332_inst = (S1232 & S1332) | (S1332 & S1432) | (S1232 &
    S1432);
699 assign S2333 = S1333 ^ S1433;
700 assign C2333_inst = S1333 & S1433;
701 assign S2334 = S1334 ^ S1434;

```

```

702 assign C2334_inst = S1334 & S1434;
703 assign S2335 = S1335 ^ P22[35];
704 assign C2335_inst = S1335 & P22[35];
705
706 assign S2415 = S1515 ^ P15[15];
707 assign C2415_inst = S1515 & P15[15];
708 assign S2416 = S1516 ^ S1616;
709 assign C2416_inst = S1516 & S1616;
710 assign S2417 = S1417 ^ S1517 ^ S1617;
711 assign C2417_inst = (S1417 & S1517) | (S1517 & S1617) | (S1417 &
    S1617);
712 assign S2418 = S1418 ^ S1518 ^ S1618;
713 assign C2418_inst = (S1418 & S1518) | (S1518 & S1618) | (S1418 &
    S1618);
714 assign S2419 = S1419 ^ S1519 ^ S1619;
715 assign C2419_inst = (S1419 & S1519) | (S1519 & S1619) | (S1419 &
    S1619);
716 assign S2420 = S1320 ^ S1420 ^ S1520;
717 assign C2420_inst = (S1320 & S1420) | (S1420 & S1520) | (S1320 &
    S1520);
718 assign S2421 = S1321 ^ S1421 ^ S1521;
719 assign C2421_inst = (S1321 & S1421) | (S1421 & S1521) | (S1321 &
    S1521);
720 assign S2422 = S1322 ^ S1422 ^ S1522;
721 assign C2422_inst = (S1322 & S1422) | (S1422 & S1522) | (S1322 &
    S1522);
722 assign S2423 = S1223 ^ S1323 ^ S1423;
723 assign C2423_inst = (S1223 & S1323) | (S1323 & S1423) | (S1223 &
    S1423);
724 assign S2424 = S1324 ^ S1424 ^ S1524;
725 assign C2424_inst = (S1324 & S1424) | (S1424 & S1524) | (S1324 &
    S1524);
726 assign S2425 = S1325 ^ S1425 ^ S1525;
727 assign C2425_inst = (S1325 & S1425) | (S1425 & S1525) | (S1325 &
    S1525);
728 assign S2426 = S1326 ^ S1426 ^ S1526;
729 assign C2426_inst = (S1326 & S1426) | (S1426 & S1526) | (S1326 &
    S1526);
730 assign S2427 = S1427 ^ S1527 ^ S1627;
731 assign C2427_inst = (S1427 & S1527) | (S1527 & S1627) | (S1427 &
    S1627);

```

```

732 assign S2428 = S1428 ^ S1528 ^ S1628;
733 assign C2428_inst = (S1428 & S1528) | (S1528 & S1628) | (S1428 &
    S1628);
734 assign S2429 = S1429 ^ S1529 ^ P22[29];
735 assign C2429_inst = (S1429 & S1529) | (S1529 & P22[29]) | (S1429
    & P22[29]);
736
737 assign S2520 = S1620 ^ S1720;
738 assign C2520_inst = S1620 & S1720;
739 assign S2521 = S1621 ^ S1721 ^ P21[21];
740 assign C2521_inst = (S1621 & S1721) | (S1721 & P21[21]) | (S1621
    & P21[21]);
741 assign S2522 = S1622 ^ S1722 ^ S1822;
742 assign C2522_inst = (S1622 & S1722) | (S1722 & S1822) | (S1622 &
    S1822);
743 assign S2523 = S1523 ^ S1623 ^ S1723;
744 assign C2523_inst = (S1523 & S1623) | (S1623 & S1723) | (S1523 &
    S1723);
745 assign S2524 = S1624 ^ S1724;
746 assign C2524_inst = S1624 & S1724;
747 assign S2525 = S1625 ^ S1725;
748 assign C2525_inst = S1625 & S1725;
749 assign S2526 = S1626 ^ P22[26];
750 assign C2526_inst = S1626 & P22[26];
751
752 //Third Stage_inst
753
754 assign S33 = S23 ^ C22;
755 assign C33 = S23 & C22;
756 assign S35 = S25 ^ S125 ^ C24;
757 assign C35 = (S25 & S125) | (S125 & C24) | (S25 & C24);
758 assign S36 = S216 ^ S226 ^ C25;
759 assign C36 = (S216 & S226) | (S226 & C25) | (S216 & C25);
760 assign S37 = S217 ^ S227 ^ C216;
761 assign C37 = (S217 & S227) | (S227 & C216) | (S217 & C216);
762 assign S38 = S218 ^ S228 ^ C217;
763 assign C38 = (S218 & S228) | (S228 & C217) | (S218 & C217);
764 assign S319 = S219 ^ S229 ^ C218;
765 assign C319 = (S219 & S229) | (S229 & C218) | (S219 & C218);
766 assign S3110 = S2110 ^ S2210 ^ C219;

```

```

767 assign C3110_inst = (S2110 & S2210) | (S2210 & C219) | (S2110 &
      C219);
768 assign S3111 = S2111 ^ S2211 ^ S2311;
769 assign C3111_inst = (S2111 & S2211) | (S2211 & S2311) | (S2111 &
      S2311);
770 assign S3112 = S2112 ^ S2212 ^ S2312;
771 assign C3112_inst = (S2112 & S2212) | (S2212 & S2312) | (S2112 &
      S2312);
772 assign S3113 = S2113 ^ S2213 ^ S2313;
773 assign C3113_inst = (S2113 & S2213) | (S2213 & S2313) | (S2113 &
      S2313);
774 assign S3114 = S2114 ^ S2214 ^ S2314;
775 assign C3114_inst = (S2114 & S2214) | (S2214 & S2314) | (S2114 &
      S2314);
776 assign S3115 = S2115 ^ S2215 ^ S2315;
777 assign C3115_inst = (S2115 & S2215) | (S2215 & S2315) | (S2115 &
      S2315);
778 assign S3116 = S2116 ^ S2216 ^ S2316;
779 assign C3116_inst = (S2116 & S2216) | (S2216 & S2316) | (S2116 &
      S2316);
780 assign S3117 = S2117 ^ S2217 ^ S2317;
781 assign C3117_inst = (S2117 & S2217) | (S2217 & S2317) | (S2117 &
      S2317);
782 assign S3118 = S2118 ^ S2218 ^ S2318;
783 assign C3118_inst = (S2118 & S2218) | (S2218 & S2318) | (S2118 &
      S2318);
784 assign S3119 = S2119 ^ S2219 ^ S2319;
785 assign C3119_inst = (S2119 & S2219) | (S2219 & S2319) | (S2119 &
      S2319);
786 assign S3120 = S2120 ^ S2220 ^ S2320;
787 assign C3120_inst = (S2120 & S2220) | (S2220 & S2320) | (S2120 &
      S2320);
788 assign S3121 = S2121 ^ S2221 ^ S2321;
789 assign C3121_inst = (S2121 & S2221) | (S2221 & S2321) | (S2121 &
      S2321);
790 assign S3122 = S2122 ^ S2222 ^ S2322;
791 assign C3122_inst = (S2122 & S2222) | (S2222 & S2322) | (S2122 &
      S2322);
792 assign S3123 = S2123 ^ S2223 ^ S2323;
793 assign C3123_inst = (S2123 & S2223) | (S2223 & S2323) | (S2123 &
      S2323);

```



```

794 assign S3124 = S2124 ^ S2224 ^ S2324;
795 assign C3124_inst = (S2124 & S2224) | (S2224 & S2324) | (S2124 &
    S2324);
796 assign S3125 = S2125 ^ S2225 ^ S2325;
797 assign C3125_inst = (S2125 & S2225) | (S2225 & S2325) | (S2125 &
    S2325);
798 assign S3126 = S2126 ^ S2226 ^ S2326;
799 assign C3126_inst = (S2126 & S2226) | (S2226 & S2326) | (S2126 &
    S2326);
800 assign S3127 = S2127 ^ S2227 ^ S2327;
801 assign C3127_inst = (S2127 & S2227) | (S2227 & S2327) | (S2127 &
    S2327);
802 assign S3128 = S2128 ^ S2228 ^ S2328;
803 assign C3128_inst = (S2128 & S2228) | (S2228 & S2328) | (S2128 &
    S2328);
804 assign S3129 = S2129 ^ S2229 ^ S2329;
805 assign C3129_inst = (S2129 & S2229) | (S2229 & S2329) | (S2129 &
    S2329);
806 assign S3130 = S2130 ^ S2230 ^ S2330;
807 assign C3130_inst = (S2130 & S2230) | (S2230 & S2330) | (S2130 &
    S2330);
808 assign S3131 = S2131 ^ S2231 ^ S2331;
809 assign C3131_inst = (S2131 & S2231) | (S2231 & S2331) | (S2131 &
    S2331);
810 assign S3132 = S2132 ^ S2232 ^ S2332;
811 assign C3132_inst = (S2132 & S2232) | (S2232 & S2332) | (S2132 &
    S2332);
812 assign S3133 = S2133 ^ S2233 ^ S2333;
813 assign C3133_inst = (S2133 & S2233) | (S2233 & S2333) | (S2133 &
    S2333);
814 assign S3134 = S2134 ^ S2234 ^ S2334;
815 assign C3134_inst = (S2134 & S2234) | (S2234 & S2334) | (S2134 &
    S2334);
816 assign S3135 = S2135 ^ S2235 ^ S2335;
817 assign C3135_inst = (S2135 & S2235) | (S2235 & S2335) | (S2135 &
    S2335);
818 assign S3136 = S2136 ^ S2236 ^ C2135_inst;
819 assign C3136_inst = (S2136 & S2236) | (S2236 & C2135_inst) | (
    S2136 & C2135_inst);
820 assign S3137 = S2137 ^ S2237 ^ C2136_inst;

```

```

821 assign C3137_inst = (S2137 & S2237) | (S2237 & C2136_inst) | (
      S2137 & C2136_inst);
822 assign S3138 = S2138 ^ S2238 ^ C2137_inst;
823 assign C3138_inst = (S2138 & S2238) | (S2238 & C2137_inst) | (
      S2138 & C2137_inst);
824 assign S3139 = S2139 ^ C2138_inst ^ C2238_inst;
825 assign C3139_inst = (S2139 & C2138_inst) | (C2138_inst &
      C2238_inst) | (S2139 & C2238_inst);
826 assign S3140 = S2140 ^ C2139_inst ^ S1240;
827 assign C3140_inst = (S2140 & C2139_inst) | (C2139_inst & S1240) |
      (S2140 & S1240);
828 assign S3141 = S2141 ^ C2140_inst ^ P22[41];
829 assign C3141_inst = (S2141 & C2140_inst) | (C2140_inst & P22[41])
      | (S2141 & P22[41]);
830 assign S3142 = S1142 ^ C1141_inst ^ C2141_inst;
831 assign C3142_inst = (S1142 & C1141_inst) | (C1141_inst &
      C2141_inst) | (S1142 & C2141_inst);
832 assign S3143 = S1143 ^ C1142_inst ^ C3142_inst;
833 assign C3143_inst = (S1143 & C1142_inst) | (C1142_inst &
      C3142_inst) | (S1143 & C3142_inst);
834 assign S3144 = P22[44] ^ C1143_inst ^ C3143_inst;
835 assign C3144_inst = (P22[44] & C1143_inst) | (C1143_inst &
      C3143_inst) | (P22[44] & C3143_inst);
836
837 assign S329 = C228 ^ P9[9];
838 assign C329 = C228 & P9[9];
839 assign S3210 = C229 ^ S1410;
840 assign C3210_inst = C229 & S1410;
841 assign S3211 = C2110_inst ^ C2210_inst;
842 assign C3211_inst = C2110_inst & C2210_inst;
843 assign S3212 = C2111_inst ^ C2211_inst ^ C2311_inst;
844 assign C3212_inst = (C2111_inst & C2211_inst) | (C2211_inst &
      C2311_inst) | (C2111_inst & C2311_inst);
845 assign S3213 = C2112_inst ^ C2212_inst ^ C2312_inst;
846 assign C3213_inst = (C2112_inst & C2212_inst) | (C2212_inst &
      C2312_inst) | (C2112_inst & C2312_inst);
847 assign S3214 = C2113_inst ^ C2213_inst ^ C2313_inst;
848 assign C3214_inst = (C2113_inst & C2213_inst) | (C2213_inst &
      C2313_inst) | (C2113_inst & C2313_inst);
849 assign S3215 = S2415 ^ C2114_inst ^ C2214_inst;

```

```

850 assign C3215_inst = (S2415 & C2114_inst) | (C2114_inst &
      C2214_inst) | (S2415 & C2214_inst);
851 assign S3216 = S2416 ^ C2115_inst ^ C2215_inst;
852 assign C3216_inst = (S2416 & C2115_inst) | (C2115_inst &
      C2215_inst) | (S2416 & C2215_inst);
853 assign S3217 = S2417 ^ C2116_inst ^ C2216_inst;
854 assign C3217_inst = (S2417 & C2116_inst) | (C2116_inst &
      C2216_inst) | (S2417 & C2216_inst);
855 assign S3218 = S2418 ^ C2117_inst ^ C2217_inst;
856 assign C3218_inst = (S2418 & C2117_inst) | (C2117_inst &
      C2217_inst) | (S2418 & C2217_inst);
857 assign S3219 = S2419 ^ C2118_inst ^ C2218_inst;
858 assign C3219_inst = (S2419 & C2118_inst) | (C2118_inst &
      C2218_inst) | (S2419 & C2218_inst);
859 assign S3220 = S2420 ^ S2520 ^ C2119_inst;
860 assign C3220_inst = (S2420 & S2520) | (S2520 & C2119_inst) | (
      S2420 & C2119_inst);
861 assign S3221 = S2421 ^ S2521 ^ C2120_inst;
862 assign C3221_inst = (S2421 & S2521) | (S2521 & C2120_inst) | (
      S2421 & C2120_inst);
863 assign S3222 = S2422 ^ S2522 ^ C2121_inst;
864 assign C3222_inst = (S2422 & S2522) | (S2522 & C2121_inst) | (
      S2422 & C2121_inst);
865 assign S3223 = S2423 ^ S2523 ^ C2122_inst;
866 assign C3223_inst = (S2423 & S2523) | (S2523 & C2122_inst) | (
      S2423 & C2122_inst);
867 assign S3224 = S2424 ^ S2524 ^ C2123_inst;
868 assign C3224_inst = (S2424 & S2524) | (S2524 & C2123_inst) | (
      S2424 & C2123_inst);
869 assign S3225 = S2425 ^ S2525 ^ C2124_inst;
870 assign C3225_inst = (S2425 & S2525) | (S2525 & C2124_inst) | (
      S2425 & C2124_inst);
871 assign S3226 = S2426 ^ S2526 ^ C2125_inst;
872 assign C3226_inst = (S2426 & S2526) | (S2526 & C2125_inst) | (
      S2426 & C2125_inst);
873 assign S3227 = S2427 ^ C2126_inst ^ C2226_inst;
874 assign C3227_inst = (S2427 & C2126_inst) | (C2126_inst &
      C2226_inst) | (S2427 & C2226_inst);
875 assign S3228 = S2428 ^ C2127_inst ^ C2227_inst;
876 assign C3228_inst = (S2428 & C2127_inst) | (C2127_inst &
      C2227_inst) | (S2428 & C2227_inst);

```

```

877 assign S3229 = S2429 ^ C2128_inst ^ C2228_inst;
878 assign C3229_inst = (S2429 & C2128_inst) | (C2128_inst &
      C2228_inst) | (S2429 & C2228_inst);
879 assign S3230 = C2129_inst ^ C2229_inst ^ C2329_inst;
880 assign C3230_inst = (C2129_inst & C2229_inst) | (C2229_inst &
      C2329_inst) | (C2129_inst & C2329_inst);
881 assign S3231 = C2130_inst ^ C2230_inst ^ C2330_inst;
882 assign C3231_inst = (C2130_inst & C2230_inst) | (C2230_inst &
      C2330_inst) | (C2130_inst & C2330_inst);
883 assign S3232 = C2131_inst ^ C2231_inst ^ C2331_inst;
884 assign C3232_inst = (C2131_inst & C2231_inst) | (C2231_inst &
      C2331_inst) | (C2131_inst & C2331_inst);
885 assign S3233 = C2132_inst ^ C2232_inst ^ C2332_inst;
886 assign C3233_inst = (C2132_inst & C2232_inst) | (C2232_inst &
      C2332_inst) | (C2132_inst & C2332_inst);
887 assign S3234 = C2133_inst ^ C2233_inst ^ C2333_inst;
888 assign C3234_inst = (C2133_inst & C2233_inst) | (C2233_inst &
      C2333_inst) | (C2133_inst & C2333_inst);
889 assign S3235 = C2134_inst ^ C2234_inst ^ C2334_inst;
890 assign C3235_inst = (C2134_inst & C2234_inst) | (C2234_inst &
      C2334_inst) | (C2134_inst & C2334_inst);
891 assign S3236 = C2235_inst ^ C2335_inst;
892 assign C3236_inst = C2235_inst & C2335_inst;
893
894 assign S3316 = C2315_inst ^ C2415_inst;
895 assign C3316_inst = C2315_inst & C2415_inst;
896 assign S3317 = C2316_inst ^ C2416_inst;
897 assign C3317_inst = C2316_inst & C2416_inst;
898 assign S3318 = C2317_inst ^ C2417_inst ^ P18[18];
899 assign C3318_inst = (C2317_inst & C2417_inst) | (C2417_inst & P18
      [18]) | (C2317_inst & P18[18]);
900 assign S3319 = C2318_inst ^ C2418_inst ^ S1719;
901 assign C3319_inst = (C2318_inst & C2418_inst) | (C2418_inst &
      S1719) | (C2318_inst & S1719);
902 assign S3320 = C2219_inst ^ C2319_inst ^ C2419_inst;
903 assign C3320_inst = (C2219_inst & C2319_inst) | (C2319_inst &
      C2419_inst) | (C2219_inst & C2419_inst);
904 assign S3321 = C2220_inst ^ C2320_inst ^ C2420_inst;
905 assign C3321_inst = (C2220_inst & C2320_inst) | (C2320_inst &
      C2420_inst) | (C2220_inst & C2420_inst);
906 assign S3322 = C2221_inst ^ C2321_inst ^ C2421_inst;

```

```

907 assign C3322_inst = (C2221_inst & C2321_inst) | (C2321_inst &
    C2421_inst) | (C2221_inst & C2421_inst);
908 assign S3323 = C2222_inst ^ C2322_inst ^ C2422_inst;
909 assign C3323_inst = (C2222_inst & C2322_inst) | (C2322_inst &
    C2422_inst) | (C2222_inst & C2422_inst);
910 assign S3324 = C2223_inst ^ C2323_inst ^ C2423_inst;
911 assign C3324_inst = (C2223_inst & C2323_inst) | (C2323_inst &
    C2423_inst) | (C2223_inst & C2423_inst);
912 assign S3325 = C2224_inst ^ C2324_inst ^ C2424_inst;
913 assign C3325_inst = (C2224_inst & C2324_inst) | (C2324_inst &
    C2424_inst) | (C2224_inst & C2424_inst);
914 assign S3326 = C2225_inst ^ C2325_inst ^ C2425_inst;
915 assign C3326_inst = (C2225_inst & C2325_inst) | (C2325_inst &
    C2425_inst) | (C2225_inst & C2425_inst);
916 assign S3327 = C2326_inst ^ C2426_inst ^ C2526_inst;
917 assign C3327_inst = (C2326_inst & C2426_inst) | (C2426_inst &
    C2526_inst) | (C2326_inst & C2526_inst);
918 assign S3328 = C2327_inst ^ C2427_inst;
919 assign C3328_inst = C2327_inst & C2427_inst;
920 assign S3329 = C2328_inst ^ C2428_inst;
921 assign C3329_inst = C2328_inst & C2428_inst;
922 assign S3330 = C2429_inst ^ S1530;
923 assign C3330_inst = C2429_inst & S1530;
924
925 assign S3423 = C2522_inst ^ P22[23];
926 assign C3423_inst = C2522_inst & P22[23];
927
928 //Fourth Stage
929
930 assign S44 = S24 ^ C23 ^ C33;
931 assign C44 = (S24 & C23) | (C23 & C33) | (S24 & C33);
932 assign S45 = S35 ^ C44;
933 assign C45 = S35 & C44;
934 assign S46 = S36 ^ C35 ^ C45;
935 assign C46 = (S36 & C35) | (C35 & C45) | (S36 & C45);
936 assign S47 = S37 ^ C36 ^ C226;
937 assign C47 = (S37 & C36) | (C36 & C226) | (S37 & C226);
938 assign S48 = S38 ^ C37 ^ C227;
939 assign C48 = (S38 & C37) | (C37 & C227) | (S38 & C227);
940 assign S49 = S319 ^ S329 ^ C38;
941 assign C49 = (S319 & S329) | (S329 & C38) | (S319 & C38);

```

```

942 assign S410 = S3110 ^ S3210 ^ C319;
943 assign C410 = (S3110 & S3210) | (S3210 & C319) | (S3110 & C319);
944 assign S411 = S3111 ^ S3211 ^ C3110_inst;
945 assign C411 = (S3111 & S3211) | (S3211 & C3110_inst) | (S3111 &
    C3110_inst);
946 assign S412 = S3112 ^ S3212 ^ C3111_inst;
947 assign C412 = (S3112 & S3212) | (S3212 & C3111_inst) | (S3112 &
    C3111_inst);
948 assign S413 = S3113 ^ S3213 ^ C3112_inst;
949 assign C413 = (S3113 & S3213) | (S3213 & C3112_inst) | (S3113 &
    C3112_inst);
950 assign S4114 = S3114 ^ S3214 ^ C3113_inst;
951 assign C4114_inst = (S3114 & S3214) | (S3214 & C3113_inst) | (
    S3114 & C3113_inst);
952 assign S4115 = S3115 ^ S3215 ^ C3114_inst;
953 assign C4115_inst = (S3115 & S3215) | (S3215 & C3114_inst) | (
    S3115 & C3114_inst);
954 assign S4116 = S3116 ^ S3216 ^ S3316;
955 assign C4116_inst = (S3116 & S3216) | (S3216 & S3316) | (S3116 &
    S3316);
956 assign S4117 = S3117 ^ S3217 ^ S3317;
957 assign C4117_inst = (S3117 & S3217) | (S3217 & S3317) | (S3117 &
    S3317);
958 assign S4118 = S3118 ^ S3218 ^ S3318;
959 assign C4118_inst = (S3118 & S3218) | (S3218 & S3318) | (S3118 &
    S3318);
960 assign S4119 = S3119 ^ S3219 ^ S3319;
961 assign C4119_inst = (S3119 & S3219) | (S3219 & S3319) | (S3119 &
    S3319);
962 assign S4120 = S3120 ^ S3220 ^ S3320;
963 assign C4120_inst = (S3120 & S3220) | (S3220 & S3320) | (S3120 &
    S3320);
964 assign S4121 = S3121 ^ S3221 ^ S3321;
965 assign C4121_inst = (S3121 & S3221) | (S3221 & S3321) | (S3121 &
    S3321);
966 assign S4122 = S3122 ^ S3222 ^ S3322;
967 assign C4122_inst = (S3122 & S3222) | (S3222 & S3322) | (S3122 &
    S3322);
968 assign S4123 = S3123 ^ S3223 ^ S3323;
969 assign C4123_inst = (S3123 & S3223) | (S3223 & S3323) | (S3123 &
    S3323);

```

```

970 assign S4124 = S3124 ^ S3224 ^ S3324;
971 assign C4124_inst = (S3124 & S3224) | (S3224 & S3324) | (S3124 &
    S3324);
972 assign S4125 = S3125 ^ S3225 ^ S3325;
973 assign C4125_inst = (S3125 & S3225) | (S3225 & S3325) | (S3125 &
    S3325);
974 assign S4126 = S3126 ^ S3226 ^ S3326;
975 assign C4126_inst = (S3126 & S3226) | (S3226 & S3326) | (S3126 &
    S3326);
976 assign S4127 = S3127 ^ S3227 ^ S3327;
977 assign C4127_inst = (S3127 & S3227) | (S3227 & S3327) | (S3127 &
    S3327);
978 assign S4128 = S3128 ^ S3228 ^ S3328;
979 assign C4128_inst = (S3128 & S3228) | (S3228 & S3328) | (S3128 &
    S3328);
980 assign S4129 = S3129 ^ S3229 ^ S3329;
981 assign C4129_inst = (S3129 & S3229) | (S3229 & S3329) | (S3129 &
    S3329);
982 assign S4130 = S3130 ^ S3230 ^ S3330;
983 assign C4130_inst = (S3130 & S3230) | (S3230 & S3330) | (S3130 &
    S3330);
984 assign S4131 = S3131 ^ S3231 ^ C3130_inst;
985 assign C4131_inst = (S3131 & S3231) | (S3231 & C3130_inst) | (
    S3131 & C3130_inst);
986 assign S4132 = S3132 ^ S3232 ^ C3131_inst;
987 assign C4132_inst = (S3132 & S3232) | (S3232 & C3131_inst) | (
    S3132 & C3131_inst);
988 assign S4133 = S3133 ^ S3233 ^ C3132_inst;
989 assign C4133_inst = (S3133 & S3233) | (S3233 & C3132_inst) | (
    S3133 & C3132_inst);
990 assign S4134 = S3134 ^ S3234 ^ C3133_inst;
991 assign C4134_inst = (S3134 & S3234) | (S3234 & C3133_inst) | (
    S3134 & C3133_inst);
992 assign S4135 = S3135 ^ S3235 ^ C3134_inst;
993 assign C4135_inst = (S3135 & S3235) | (S3235 & C3134_inst) | (
    S3135 & C3134_inst);
994 assign S4136 = S3136 ^ S3236 ^ C3135_inst;
995 assign C4136_inst = (S3136 & S3236) | (S3236 & C3135_inst) | (
    S3136 & C3135_inst);
996 assign S4137 = S3137 ^ C3136_inst ^ C3236_inst;

```

```

997 assign C4137_inst = (S3137 & C3136_inst) | (C3136_inst &
    C3236_inst) | (S3137 & C3236_inst);
998 assign S4138 = S3138 ^ C3137_inst ^ C2237_inst;
999 assign C4138_inst = (S3138 & C3137_inst) | (C3137_inst &
    C2237_inst) | (S3138 & C2237_inst);
1000 assign S4139 = S3139 ^ C3138_inst ^ S1239;
1001 assign C4139_inst = (S3139 & C3138_inst) | (C3138_inst & S1239) |
    (S3139 & S1239);
1002
1003 assign S4214 = C3213_inst ^ S1514;
1004 assign C4214_inst = C3213_inst & S1514;
1005 assign S4215 = C3214_inst ^ C2314_inst;
1006 assign C4215_inst = C3214_inst & C2314_inst;
1007 assign S4216 = C3115_inst ^ C3215_inst;
1008 assign C4216_inst = C3115_inst & C3215_inst;
1009 assign S4217 = C3116_inst ^ C3216_inst ^ C3316_inst;
1010 assign C4217_inst = (C3116_inst & C3216_inst) | (C3216_inst &
    C3316_inst) | (C3116_inst & C3316_inst);
1011 assign S4218 = C3117_inst ^ C3217_inst ^ C3317_inst;
1012 assign C4218_inst = (C3117_inst & C3217_inst) | (C3217_inst &
    C3317_inst) | (C3117_inst & C3317_inst);
1013 assign S4219 = C3118_inst ^ C3218_inst ^ C3318_inst;
1014 assign C4219_inst = (C3118_inst & C3218_inst) | (C3218_inst &
    C3318_inst) | (C3118_inst & C3318_inst);
1015 assign S4220 = C3119_inst ^ C3219_inst ^ C3319_inst;
1016 assign C4220_inst = (C3119_inst & C3219_inst) | (C3219_inst &
    C3319_inst) | (C3119_inst & C3319_inst);
1017 assign S4221 = C3120_inst ^ C3220_inst ^ C3320_inst;
1018 assign C4221_inst = (C3120_inst & C3220_inst) | (C3220_inst &
    C3320_inst) | (C3120_inst & C3320_inst);
1019 assign S4222 = C3121_inst ^ C3221_inst ^ C3321_inst;
1020 assign C4222_inst = (C3121_inst & C3221_inst) | (C3221_inst &
    C3321_inst) | (C3121_inst & C3321_inst);
1021 assign S4223 = S3423 ^ C3122_inst ^ C3222_inst;
1022 assign C4223_inst = (S3423 & C3122_inst) | (C3122_inst &
    C3222_inst) | (S3423 & C3222_inst);
1023 assign S4224 = C3123_inst ^ C3223_inst ^ C3323_inst;
1024 assign C4224_inst = (C3123_inst & C3223_inst) | (C3223_inst &
    C3323_inst) | (C3123_inst & C3323_inst);
1025 assign S4225 = C3124_inst ^ C3224_inst ^ C3324_inst;

```



```

1026 assign C4225_inst = (C3124_inst & C3224_inst) | (C3224_inst &
      C3324_inst) | (C3124_inst & C3324_inst);
1027 assign S4226 = C3125_inst ^ C3225_inst ^ C3325_inst;
1028 assign C4226_inst = (C3125_inst & C3225_inst) | (C3225_inst &
      C3325_inst) | (C3125_inst & C3325_inst);
1029 assign S4227 = C3126_inst ^ C3226_inst ^ C3326_inst;
1030 assign C4227_inst = (C3126_inst & C3226_inst) | (C3226_inst &
      C3326_inst) | (C3126_inst & C3326_inst);
1031 assign S4228 = C3127_inst ^ C3227_inst ^ C3327_inst;
1032 assign C4228_inst = (C3127_inst & C3227_inst) | (C3227_inst &
      C3327_inst) | (C3127_inst & C3327_inst);
1033 assign S4229 = C3128_inst ^ C3228_inst ^ C3328_inst;
1034 assign C4229_inst = (C3128_inst & C3228_inst) | (C3228_inst &
      C3328_inst) | (C3128_inst & C3328_inst);
1035 assign S4230 = C3129_inst ^ C3229_inst ^ C3329_inst;
1036 assign C4230_inst = (C3129_inst & C3229_inst) | (C3229_inst &
      C3329_inst) | (C3129_inst & C3329_inst);
1037 assign S4231 = C3230_inst ^ C3330_inst ^ S1531;
1038 assign C4231_inst = (C3230_inst & C3330_inst) | (C3330_inst &
      S1531) | (C3230_inst & S1531);
1039 assign S4232 = C3231_inst ^ P22[32];
1040 assign C4232_inst = C3231_inst & P22[32];
1041
1042 assign S4324 = C3423_inst ^ C2523_inst;
1043 assign C4324_inst = C3423_inst & C2523_inst;
1044
1045 //Fifth Stage
1046
1047 assign S57 = S47 ^ C46;
1048 assign C57 = S47 & C46;
1049 assign S58 = S48 ^ C47 ^ C57;
1050 assign C58 = (S48 & C47) | (C47 & C57) | (S48 & C57);
1051 assign S59 = S49 ^ C48 ^ C58;
1052 assign C59 = (S49 & C48) | (C48 & C58) | (S49 & C58);
1053 assign S510 = S410 ^ C49 ^ C329;
1054 assign C510 = (S410 & C49) | (C49 & C329) | (S410 & C329);
1055 assign S511 = S411 ^ C410 ^ C3210_inst;
1056 assign C511 = (S411 & C410) | (C410 & C3210_inst) | (S411 &
      C3210_inst);
1057 assign S512 = S412 ^ C411 ^ C3211_inst;

```

```

1058 assign C512 = (S412 & C411) | (C411 & C3211_inst) | (S412 &
      C3211_inst);
1059 assign S513 = S413 ^ C412 ^ C3212_inst;
1060 assign C513 = (S413 & C412) | (C412 & C3212_inst) | (S413 &
      C3212_inst);
1061 assign S514 = S4114 ^ S4214 ^ C413;
1062 assign C514 = (S4114 & S4214) | (S4214 & C413) | (S4114 & C413);
1063 assign S515 = S4115 ^ S4215 ^ C4114_inst;
1064 assign C515 = (S4115 & S4215) | (S4215 & C4114_inst) | (S4115 &
      C4114_inst);
1065 assign S516 = S4116 ^ S4216 ^ C4115_inst;
1066 assign C516 = (S4116 & S4216) | (S4216 & C4115_inst) | (S4116 &
      C4115_inst);
1067 assign S517 = S4117 ^ S4217 ^ C4116_inst;
1068 assign C517 = (S4117 & S4217) | (S4217 & C4116_inst) | (S4117 &
      C4116_inst);
1069 assign S518 = S4118 ^ S4218 ^ C4117_inst;
1070 assign C518 = (S4118 & S4218) | (S4218 & C4117_inst) | (S4118 &
      C4117_inst);
1071 assign S519 = S4119 ^ S4219 ^ C4118_inst;
1072 assign C519 = (S4119 & S4219) | (S4219 & C4118_inst) | (S4119 &
      C4118_inst);
1073 assign S520 = S4120 ^ S4220 ^ C4119_inst;
1074 assign C520 = (S4120 & S4220) | (S4220 & C4119_inst) | (S4120 &
      C4119_inst);
1075 assign S5121 = S4121 ^ S4221 ^ C4120_inst;
1076 assign C5121_inst = (S4121 & S4221) | (S4221 & C4120_inst) | (
      S4121 & C4120_inst);
1077 assign S5122 = S4122 ^ S4222 ^ C4121_inst;
1078 assign C5122_inst = (S4122 & S4222) | (S4222 & C4121_inst) | (
      S4122 & C4121_inst);
1079 assign S5123 = S4123 ^ S4223 ^ C4122_inst;
1080 assign C5123_inst = (S4123 & S4223) | (S4223 & C4122_inst) | (
      S4123 & C4122_inst);
1081 assign S5124 = S4124 ^ S4224 ^ S4324;
1082 assign C5124_inst = (S4124 & S4224) | (S4224 & S4324) | (S4124 &
      S4324);
1083 assign S5125 = S4125 ^ S4225 ^ C4124_inst;
1084 assign C5125_inst = (S4125 & S4225) | (S4225 & C4124_inst) | (
      S4125 & C4124_inst);
1085 assign S5126 = S4126 ^ S4226 ^ C4125_inst;

```

```

1086 assign C5126_inst = (S4126 & S4226) | (S4226 & C4125_inst) | (
      S4126 & C4125_inst);
1087 assign S5127 = S4127 ^ S4227 ^ C4126_inst;
1088 assign C5127_inst = (S4127 & S4227) | (S4227 & C4126_inst) | (
      S4127 & C4126_inst);
1089 assign S5128 = S4128 ^ S4228 ^ C4127_inst;
1090 assign C5128_inst = (S4128 & S4228) | (S4228 & C4127_inst) | (
      S4128 & C4127_inst);
1091 assign S5129 = S4129 ^ S4229 ^ C4128_inst;
1092 assign C5129_inst = (S4129 & S4229) | (S4229 & C4128_inst) | (
      S4129 & C4128_inst);
1093 assign S5130 = S4130 ^ S4230 ^ C4129_inst;
1094 assign C5130_inst = (S4130 & S4230) | (S4230 & C4129_inst) | (
      S4130 & C4129_inst);
1095 assign S5131 = S4131 ^ S4231 ^ C4130_inst;
1096 assign C5131_inst = (S4131 & S4231) | (S4231 & C4130_inst) | (
      S4131 & C4130_inst);
1097 assign S5132 = S4132 ^ S4232 ^ C4131_inst;
1098 assign C5132_inst = (S4132 & S4232) | (S4232 & C4131_inst) | (
      S4132 & C4131_inst);
1099 assign S5133 = S4133 ^ C4132_inst ^ C4232_inst;
1100 assign C5133_inst = (S4133 & C4132_inst) | (C4132_inst &
      C4232_inst) | (S4133 & C4232_inst);
1101 assign S5134 = S4134 ^ C4133_inst ^ C3233_inst;
1102 assign C5134_inst = (S4134 & C4133_inst) | (C4133_inst &
      C3233_inst) | (S4134 & C3233_inst);
1103 assign S5135 = S4135 ^ C4134_inst ^ C3234_inst;
1104 assign C5135_inst = (S4135 & C4134_inst) | (C4134_inst &
      C3234_inst) | (S4135 & C3234_inst);
1105 assign S5136 = S4136 ^ C4135_inst ^ C3235_inst;
1106 assign C5136_inst = (S4136 & C4135_inst) | (C4135_inst &
      C3235_inst) | (S4136 & C3235_inst);
1107 assign S5137 = S4137 ^ C4136_inst ^ C2236_inst;
1108 assign C5137_inst = (S4137 & C4136_inst) | (C4136_inst &
      C2236_inst) | (S4137 & C2236_inst);
1109
1110 assign S5221 = C4220_inst ^ C2520_inst;
1111 assign C5221_inst = C4220_inst & C2520_inst;
1112 assign S5222 = C4221_inst ^ C2521_inst;
1113 assign C5222_inst = C4221_inst & C2521_inst;
1114 assign S5223 = C4222_inst ^ C3322_inst;

```

```

1115 assign C5223_inst = C4222_inst & C3322_inst;
1116 assign S5224 = C4123_inst ^ C4223_inst;
1117 assign C5224_inst = C4123_inst & C4223_inst;
1118 assign S5225 = C4224_inst ^ C4324_inst ^ C2524_inst;
1119 assign C5225_inst = (C4224_inst & C4324_inst) | (C4324_inst &
    C2524_inst) | (C4224_inst & C2524_inst);
1120 assign S5226 = C4225_inst ^ C2525_inst;
1121 assign C5226_inst = C4225_inst & C2525_inst;
1122
1123 //Sixth Stage
1124
1125 assign S610 = S510 ^ C59;
1126 assign C610 = S510 & C59;
1127 assign S611 = S511 ^ C510 ^ C610;
1128 assign C611 = (S511 & C510) | (C510 & C610) | (S511 & C610);
1129 assign S612 = S512 ^ C511 ^ C611;
1130 assign C612 = (S512 & C511) | (C511 & C611) | (S512 & C611);
1131 assign S613 = S513 ^ C512 ^ C612;
1132 assign C613 = (S513 & C512) | (C512 & C612) | (S513 & C612);
1133 assign S614 = S514 ^ C513 ^ C613;
1134 assign C614 = (S514 & C513) | (C513 & C613) | (S514 & C613);
1135 assign S615 = S515 ^ C514 ^ C4214_inst;
1136 assign C615 = (S515 & C514) | (C514 & C4214_inst) | (S515 &
    C4214_inst);
1137 assign S616 = S516 ^ C515 ^ C4215_inst;
1138 assign C616 = (S516 & C515) | (C515 & C4215_inst) | (S516 &
    C4215_inst);
1139 assign S617 = S517 ^ C516 ^ C4216_inst;
1140 assign C617 = (S517 & C516) | (C516 & C4216_inst) | (S517 &
    C4216_inst);
1141 assign S618 = S518 ^ C517 ^ C4217_inst;
1142 assign C618 = (S518 & C517) | (C517 & C4217_inst) | (S518 &
    C4217_inst);
1143 assign S619 = S519 ^ C518 ^ C4218_inst;
1144 assign C619 = (S519 & C518) | (C518 & C4218_inst) | (S519 &
    C4218_inst);
1145 assign S620 = S520 ^ C519 ^ C4219_inst;
1146 assign C620 = (S520 & C519) | (C519 & C4219_inst) | (S520 &
    C4219_inst);
1147 assign S621 = S5121 ^ S5221 ^ C520;
1148 assign C621 = (S5121 & S5221) | (S5221 & C520) | (S5121 & C520);

```

```

1149 assign S622 = S5122 ^ S5222 ^ C5121_inst;
1150 assign C622 = (S5122 & S5222) | (S5222 & C5121_inst) | (S5122 &
      C5121_inst);
1151 assign S623 = S5123 ^ S5223 ^ C5122_inst;
1152 assign C623 = (S5123 & S5223) | (S5223 & C5122_inst) | (S5123 &
      C5122_inst);
1153 assign S624 = S5124 ^ S5224 ^ C5123_inst;
1154 assign C624 = (S5124 & S5224) | (S5224 & C5123_inst) | (S5124 &
      C5123_inst);
1155 assign S625 = S5125 ^ S5225 ^ C5124_inst;
1156 assign C625 = (S5125 & S5225) | (S5225 & C5124_inst) | (S5125 &
      C5124_inst);
1157 assign S626 = S5126 ^ S5226 ^ C5125_inst;
1158 assign C626 = (S5126 & S5226) | (S5226 & C5125_inst) | (S5126 &
      C5125_inst);
1159 assign S627 = S5127 ^ C5126_inst ^ C5226_inst;
1160 assign C627 = (S5127 & C5126_inst) | (C5126_inst & C5226_inst) |
      (S5127 & C5226_inst);
1161 assign S628 = S5128 ^ C5127_inst ^ C4227_inst;
1162 assign C628 = (S5128 & C5127_inst) | (C5127_inst & C4227_inst) |
      (S5128 & C4227_inst);
1163 assign S629 = S5129 ^ C5128_inst ^ C4228_inst;
1164 assign C629 = (S5129 & C5128_inst) | (C5128_inst & C4228_inst) |
      (S5129 & C4228_inst);
1165 assign S630 = S5130 ^ C5129_inst ^ C4229_inst;
1166 assign C630 = (S5130 & C5129_inst) | (C5129_inst & C4229_inst) |
      (S5130 & C4229_inst);
1167 assign S631 = S5131 ^ C5130_inst ^ C4230_inst;
1168 assign C631 = (S5131 & C5130_inst) | (C5130_inst & C4230_inst) |
      (S5131 & C4230_inst);
1169 assign S632 = S5132 ^ C5131_inst ^ C4231_inst;
1170 assign C632 = (S5132 & C5131_inst) | (C5131_inst & C4231_inst) |
      (S5132 & C4231_inst);
1171 assign S633 = S5133 ^ C5132_inst ^ C3232_inst;
1172 assign C633 = (S5133 & C5132_inst) | (C5132_inst & C3232_inst) |
      (S5133 & C3232_inst);
1173 assign S634 = S5134 ^ C5133_inst ^ C633;
1174 assign C634 = (S5134 & C5133_inst) | (C5133_inst & C633) | (S5134
      & C633);
1175 assign S635 = S5135 ^ C5134_inst ^ C634;

```

```

1176 assign C635 = (S5135 & C5134_inst) | (C5134_inst & C634) | (S5135
      & C634);
1177 assign S636 = S5136 ^ C5135_inst ^ C635;
1178 assign C636 = (S5136 & C5135_inst) | (C5135_inst & C635) | (S5136
      & C635);
1179 assign S637 = S5137 ^ C5136_inst ^ C636;
1180 assign C637 = (S5137 & C5136_inst) | (C5136_inst & C636) | (S5137
      & C636);
1181 assign S638 = S4138 ^ C5137_inst ^ C637;
1182 assign C638 = (S4138 & C5137_inst) | (C5137_inst & C637) | (S4138
      & C637);
1183 assign S639 = S4139 ^ C4138_inst ^ C638;
1184 assign C639 = (S4139 & C4138_inst) | (C4138_inst & C638) | (S4139
      & C638);
1185 assign S640 = S3140 ^ C4139_inst ^ C639;
1186 assign C640 = (S3140 & C4139_inst) | (C4139_inst & C639) | (S3140
      & C639);
1187 assign S641 = S3141 ^ C3140_inst ^ C640;
1188 assign C641 = (S3141 & C3140_inst) | (C3140_inst & C640) | (S3141
      & C640);
1189 assign S642 = S3142 ^ C3141_inst ^ C641;
1190 assign C642 = (S3142 & C3141_inst) | (C3141_inst & C641) | (S3142
      & C641);
1191 assign S643 = S3143 ^ C642;
1192 assign C643 = S3143 & C642;
1193 assign S644 = S3144 ^ C643;
1194 assign C644 = S3144 & C643;
1195
1196 //Seventh Stage
1197
1198 assign S715 = S615 ^ C614;
1199 assign C715 = S615 & C614;
1200 assign S716 = S616 ^ C615 ^ C715;
1201 assign C716 = (S616 & C615) | (C615 & C715) | (S616 & C715);
1202 assign S717 = S617 ^ C616 ^ C716;
1203 assign C717 = (S617 & C616) | (C616 & C716) | (S617 & C716);
1204 assign S718 = S618 ^ C617 ^ C717;
1205 assign C718 = (S618 & C617) | (C617 & C717) | (S618 & C717);
1206 assign S719 = S619 ^ C618 ^ C718;
1207 assign C719 = (S619 & C618) | (C618 & C718) | (S619 & C718);
1208 assign S720 = S620 ^ C619 ^ C719;

```

```

1209 assign C720 = (S620 & C619) | (C619 & C719) | (S620 & C719);
1210 assign S721 = S621 ^ C620 ^ C720;
1211 assign C721 = (S621 & C620) | (C620 & C720) | (S621 & C720);
1212 assign S722 = S622 ^ C621 ^ C5221_inst;
1213 assign C722 = (S622 & C621) | (C621 & C5221_inst) | (S622 &
    C5221_inst);
1214 assign S723 = S623 ^ C622 ^ C5222_inst;
1215 assign C723 = (S623 & C622) | (C622 & C5222_inst) | (S623 &
    C5222_inst);
1216 assign S724 = S624 ^ C623 ^ C5223_inst;
1217 assign C724 = (S624 & C623) | (C623 & C5223_inst) | (S624 &
    C5223_inst);
1218 assign S725 = S625 ^ C624 ^ C5224_inst;
1219 assign C725 = (S625 & C624) | (C624 & C5224_inst) | (S625 &
    C5224_inst);
1220 assign S726 = S626 ^ C625 ^ C5225_inst;
1221 assign C726 = (S626 & C625) | (C625 & C5225_inst) | (S626 &
    C5225_inst);
1222 assign S727 = S627 ^ C626 ^ C4226_inst;
1223 assign C727 = (S627 & C626) | (C626 & C4226_inst) | (S627 &
    C4226_inst);
1224 assign S728 = S628 ^ C627;
1225 assign C728 = S628 & C627;
1226 assign S729 = S629 ^ C628 ^ C728;
1227 assign C729 = (S629 & C628) | (C628 & C728) | (S629 & C728);
1228 assign S730 = S630 ^ C629 ^ C729;
1229 assign C730 = (S630 & C629) | (C629 & C729) | (S630 & C729);
1230 assign S731 = S631 ^ C630 ^ C730;
1231 assign C731 = (S631 & C630) | (C630 & C730) | (S631 & C730);
1232 assign S732 = S632 ^ C631 ^ C731;
1233 assign C732 = (S632 & C631) | (C631 & C731) | (S632 & C731);
1234 assign S733 = S633 ^ C632 ^ C732;
1235 assign C733 = (S633 & C632) | (C632 & C732) | (S633 & C732);
1236 assign S734 = S634 ^ C733;
1237 assign C734 = S634 & C733;
1238 assign S735 = S635 ^ C734;
1239 assign C735 = S635 & C734;
1240 assign S736 = S636 ^ C735;
1241 assign C736 = S636 & C735;
1242 assign S737 = S637 ^ C736;
1243 assign C737 = S637 & C736;

```

```

1244 assign S738 = S638 ^ C737;
1245 assign C738 = S638 & C737;
1246 assign S739 = S639 ^ C738;
1247 assign C739 = S639 & C738;
1248 assign S740 = S640 ^ C739;
1249 assign C740 = S640 & C739;
1250 assign S741 = S641 ^ C740;
1251 assign C741 = S641 & C740;
1252 assign S742 = S642 ^ C741;
1253 assign C742 = S642 & C741;
1254 assign S743 = S643 ^ C742;
1255 assign C743 = S643 & C742;
1256 assign S744 = S644 ^ C743;
1257 assign C744 = S644 & C743;
1258 assign S745 = C644 ^ C744;
1259 assign C745 = C644 & C744;
1260
1261 //Eighth Stage
1262
1263 assign S822 = S722 ^ C721;
1264 assign C822 = S722 & C721;
1265 assign S823 = S723 ^ C722 ^ C822;
1266 assign C823 = (S723 & C722) | (C722 & C822) | (S723 & C822);
1267 assign S824 = S724 ^ C723 ^ C823;
1268 assign C824 = (S724 & C723) | (C723 & C823) | (S724 & C823);
1269 assign S825 = S725 ^ C724 ^ C824;
1270 assign C825 = (S725 & C724) | (C724 & C824) | (S725 & C824);
1271 assign S826 = S726 ^ C725 ^ C825;
1272 assign C826 = (S726 & C725) | (C725 & C825) | (S726 & C825);
1273 assign S827 = S727 ^ C726 ^ C826;
1274 assign C827 = (S727 & C726) | (C726 & C826) | (S727 & C826);
1275 assign S828 = S728 ^ C727 ^ C827;
1276 assign C828 = (S728 & C727) | (C727 & C827) | (S728 & C827);
1277 assign S829 = S729 ^ C828;
1278 assign C829 = S729 & C828;
1279 assign S830 = S730 ^ C829;
1280 assign C830 = S730 & C829;
1281 assign S831 = S731 ^ C830;
1282 assign C831 = S731 & C830;
1283 assign S832 = S732 ^ C831;
1284 assign C832 = S732 & C831;

```



```

1285 assign S833 = S733 ^ C832;
1286 assign C833 = S733 & C832;
1287 assign S834 = S734 ^ C833;
1288 assign C834 = S734 & C833;
1289 assign S835 = S735 ^ C834;
1290 assign C835 = S735 & C834;
1291 assign S836 = S736 ^ C835;
1292 assign C836 = S736 & C835;
1293 assign S837 = S737 ^ C836;
1294 assign C837 = S737 & C836;
1295 assign S838 = S738 ^ C837 ^ C4137_inst;
1296 assign C838 = (S738 & C837) | (C837 & C4137_inst) | (S738 &
    C4137_inst);
1297 assign S839 = S739 ^ C838;
1298 assign C839 = S739 & C838;
1299 assign S840 = S740 ^ C839 ^ C3139_inst;
1300 assign C840 = (S740 & C839) | (C839 & C3139_inst) | (S740 &
    C3139_inst);
1301 assign S841 = S741 ^ C840;
1302 assign C841 = S741 & C840;
1303 assign S842 = S742 ^ C841;
1304 assign C842 = S742 & C841;
1305 assign S843 = S743 ^ C842;
1306 assign C843 = S743 & C842;
1307 assign S844 = S744 ^ C843;
1308 assign C844 = S744 & C843;
1309 assign S845 = S745 ^ C844 ^ C3144_inst;
1310 assign C845 = (S745 & C844) | (C844 & C3144_inst) | (S745 &
    C3144_inst);
1311
1312 //Nineth Stage
1313
1314 assign S923 = S823 ^ P23[23] ^ P0[23];
1315 assign C923_inst = (S823 & P23[23]) | (P23[23] & P0[23]) | (S823
    & P0[23]);
1316 assign S924 = S824 ^ P23[24] ^ P1[24];
1317 assign C924_inst = (S824 & P23[24]) | (P23[24] & P1[24]) | (S824
    & P1[24]);
1318 assign S925 = S825 ^ P23[25] ^ P2[25];
1319 assign C925_inst = (S825 & P23[25]) | (P23[25] & P2[25]) | (S825
    & P2[25]);

```

```

1320 assign S926 = S826 ^ P23[26] ^ P3[26];
1321 assign C926_inst = (S826 & P23[26]) | (P23[26] & P3[26]) | (S826
    & P3[26]);
1322 assign S927 = S827 ^ P23[27] ^ P4[27];
1323 assign C927_inst = (S827 & P23[27]) | (P23[27] & P4[27]) | (S827
    & P4[27]);
1324 assign S928 = S828 ^ P23[28] ^ P5[28];
1325 assign C928_inst = (S828 & P23[28]) | (P23[28] & P5[28]) | (S828
    & P5[28]);
1326 assign S929 = S829 ^ P23[29] ^ P6[29];
1327 assign C929_inst = (S829 & P23[29]) | (P23[29] & P6[29]) | (S829
    & P6[29]);
1328 assign S930 = S830 ^ P23[30] ^ P7[30];
1329 assign C930_inst = (S830 & P23[30]) | (P23[30] & P7[30]) | (S830
    & P7[30]);
1330 assign S931 = S831 ^ P23[31] ^ P8[31];
1331 assign C931_inst = (S831 & P23[31]) | (P23[31] & P8[31]) | (S831
    & P8[31]);
1332 assign S932 = S832 ^ P23[32] ^ P9[32];
1333 assign C932_inst = (S832 & P23[32]) | (P23[32] & P9[32]) | (S832
    & P9[32]);
1334 assign S933 = S833 ^ P23[33] ^ P10[33];
1335 assign C933_inst = (S833 & P23[33]) | (P23[33] & P10[33]) | (S833
    & P10[33]);
1336 assign S934 = S834 ^ P23[34] ^ P11[34];
1337 assign C934_inst = (S834 & P23[34]) | (P23[34] & P11[34]) | (S834
    & P11[34]);
1338 assign S935 = S835 ^ P23[35] ^ P12[35];
1339 assign C935_inst = (S835 & P23[35]) | (P23[35] & P12[35]) | (S835
    & P12[35]);
1340 assign S936 = S836 ^ P23[36] ^ P13[36];
1341 assign C936_inst = (S836 & P23[36]) | (P23[36] & P13[36]) | (S836
    & P13[36]);
1342 assign S937 = S837 ^ P23[37] ^ P14[37];
1343 assign C937_inst = (S837 & P23[37]) | (P23[37] & P14[37]) | (S837
    & P14[37]);
1344 assign S938 = S838 ^ P23[38] ^ P15[38];
1345 assign C938_inst = (S838 & P23[38]) | (P23[38] & P15[38]) | (S838
    & P15[38]);
1346 assign S939 = S839 ^ P23[39] ^ P16[39];

```

```

1347 assign C939_inst = (S839 & P23[39]) | (P23[39] & P16[39]) | (S839
    & P16[39]);
1348 assign S940 = S840 ^ P23[40] ^ P17[40];
1349 assign C940_inst = (S840 & P23[40]) | (P23[40] & P17[40]) | (S840
    & P17[40]);
1350 assign S941 = S841 ^ P23[41] ^ P18[41];
1351 assign C941_inst = (S841 & P23[41]) | (P23[41] & P18[41]) | (S841
    & P18[41]);
1352 assign S942 = S842 ^ P23[42] ^ P19[42];
1353 assign C942_inst = (S842 & P23[42]) | (P23[42] & P19[42]) | (S842
    & P19[42]);
1354 assign S943 = S843 ^ P23[43] ^ P20[43];
1355 assign C943_inst = (S843 & P23[43]) | (P23[43] & P20[43]) | (S843
    & P20[43]);
1356 assign S944 = S844 ^ P23[44] ^ P21[44];
1357 assign C944_inst = (S844 & P23[44]) | (P23[44] & P21[44]) | (S844
    & P21[44]);
1358 assign S945 = S845 ^ P23[45] ^ P22[45];
1359 assign C945_inst = (S845 & P23[45]) | (P23[45] & P22[45]) | (S845
    & P22[45]);
1360 assign S946 = C845 ^ C745 ^ P23[46];
1361 assign C946_inst = (C845 & C745) | (C745 & P23[46]) | (C845 & P23
    [46]);
1362
1363 //Tenth Stage
1364
1365 assign S1024 = S924 ^ C923_inst;
1366 assign C1024_inst = S924 & C923_inst;
1367 assign S1025 = S925 ^ C924_inst ^ C1024_inst;
1368 assign C1025_inst = (S925 & C924_inst) | (C924_inst & C1024_inst)
    | (S925 & C1024_inst);
1369 assign S1026 = S926 ^ C925_inst ^ C1025_inst;
1370 assign C1026_inst = (S926 & C925_inst) | (C925_inst & C1025_inst)
    | (S926 & C1025_inst);
1371 assign S1027 = S927 ^ C926_inst ^ C1026_inst;
1372 assign C1027_inst = (S927 & C926_inst) | (C926_inst & C1026_inst)
    | (S927 & C1026_inst);
1373 assign S1028 = S928 ^ C927_inst ^ C1027_inst;
1374 assign C1028_inst = (S928 & C927_inst) | (C927_inst & C1027_inst)
    | (S928 & C1027_inst);
1375 assign S1029 = S929 ^ C928_inst ^ C1028_inst;

```

```

1376 assign C1029_inst = (S929 & C928_inst) | (C928_inst & C1028_inst)
      | (S929 & C1028_inst);
1377 assign S1030 = S930 ^ C929_inst ^ C1029_inst;
1378 assign C1030_inst = (S930 & C929_inst) | (C929_inst & C1029_inst)
      | (S930 & C1029_inst);
1379 assign S1031 = S931 ^ C930_inst ^ C1030_inst;
1380 assign C1031_inst = (S931 & C930_inst) | (C930_inst & C1030_inst)
      | (S931 & C1030_inst);
1381 assign S1032 = S932 ^ C931_inst ^ C1031_inst;
1382 assign C1032_inst = (S932 & C931_inst) | (C931_inst & C1031_inst)
      | (S932 & C1031_inst);
1383 assign S1033 = S933 ^ C932_inst ^ C1032_inst;
1384 assign C1033_inst = (S933 & C932_inst) | (C932_inst & C1032_inst)
      | (S933 & C1032_inst);
1385 assign S1034 = S934 ^ C933_inst ^ C1033_inst;
1386 assign C1034_inst = (S934 & C933_inst) | (C933_inst & C1033_inst)
      | (S934 & C1033_inst);
1387 assign S1035 = S935 ^ C934_inst ^ C1034_inst;
1388 assign C1035_inst = (S935 & C934_inst) | (C934_inst & C1034_inst)
      | (S935 & C1034_inst);
1389 assign S1036 = S936 ^ C935_inst ^ C1035_inst;
1390 assign C1036_inst = (S936 & C935_inst) | (C935_inst & C1035_inst)
      | (S936 & C1035_inst);
1391 assign S1037 = S937 ^ C936_inst ^ C1036_inst;
1392 assign C1037_inst = (S937 & C936_inst) | (C936_inst & C1036_inst)
      | (S937 & C1036_inst);
1393 assign S1038 = S938 ^ C937_inst ^ C1037_inst;
1394 assign C1038_inst = (S938 & C937_inst) | (C937_inst & C1037_inst)
      | (S938 & C1037_inst);
1395 assign S1039 = S939 ^ C938_inst ^ C1038_inst;
1396 assign C1039_inst = (S939 & C938_inst) | (C938_inst & C1038_inst)
      | (S939 & C1038_inst);
1397 assign S1040 = S940 ^ C939_inst ^ C1039_inst;
1398 assign C1040_inst = (S940 & C939_inst) | (C939_inst & C1039_inst)
      | (S940 & C1039_inst);
1399 assign S1041 = S941 ^ C940_inst ^ C1040_inst;
1400 assign C1041_inst = (S941 & C940_inst) | (C940_inst & C1040_inst)
      | (S941 & C1040_inst);
1401 assign S1042 = S942 ^ C941_inst ^ C1041_inst;
1402 assign C1042_inst = (S942 & C941_inst) | (C941_inst & C1041_inst)
      | (S942 & C1041_inst);

```

```

1403 assign S1043 = S943 ^ C942_inst ^ C1042_inst;
1404 assign C1043_inst = (S943 & C942_inst) | (C942_inst & C1042_inst)
      | (S943 & C1042_inst);
1405 assign S1044 = S944 ^ C943_inst ^ C1043_inst;
1406 assign C1044_inst = (S944 & C943_inst) | (C943_inst & C1043_inst)
      | (S944 & C1043_inst);
1407 assign S1045 = S945 ^ C944_inst ^ C1044_inst;
1408 assign C1045_inst = (S945 & C944_inst) | (C944_inst & C1044_inst)
      | (S945 & C1044_inst);
1409 assign S1046 = S946 ^ C945_inst ^ C1045_inst;
1410 assign C1046_inst = (S946 & C945_inst) | (C945_inst & C1045_inst)
      | (S946 & C1045_inst);
1411 assign S1047 = C946_inst ^ C1046_inst;
1412 assign C1047_inst = C946_inst & C1046_inst;
1413
1414 endmodule // mvk_mult

```

I.4 Floating Point Subtractor

```

1 module mvk_sub (
2     reset ,
3     clk ,
4     scan_in0 ,
5     scan_en ,
6     test_mode ,
7     scan_out0 ,
8     input_a ,
9     input_b ,
10    output_z
11 );
12
13 input
14     reset ,           // system reset
15     clk ;             // system clock
16
17 input
18     scan_in0 ,        // test scan mode data input
19     scan_en ,         // test scan mode enable

```

```
20      test_mode;           // test mode select
21
22  output
23      scan_out0;           // test scan mode data output
24
25  input
26      [31:0] input_a;
27
28  input
29      [31:0] input_b;
30
31  output reg
32      [31:0] output_z;
33
34  reg
35      [31:0] z_8;
36  reg
37      [26:0] a_m_1, a_m_2, b_m_1, b_m_2;
38
39  reg
40      [23:0] z_m_7, z_m_6, z_m_5, z_m_4;
41  reg
42      [9:0] a_e_1, a_e_2, b_e_1, b_e_2, z_e_7, z_e_6, z_e_5, z_e_4,
          z_e_3;
43
44  reg
45      a_s_1, a_s_2, b_s_1, b_s_2, z_s_7, z_s_6, z_s_5, z_s_4, z_s_3
          ;
46
47  reg
48      guard_6, guard_5, guard_4, round_bit_6, round_bit_5,
          round_bit_4, sticky_6, sticky_5, sticky_4;
49
50  reg
51      [27:0] sum_3;
52
53  reg
54      [8:0] abc;
55
56  reg
57      [8:0] xyz;
```

```
58
59 reg stall_mc10 , stall_mc1 , stall_mc2 , stall_mc3 , stall_mc4 ,
    stall_mc5 , stall_mc6 , stall_mc7 , stall_mc8 , stall_mc9 ;
60
61 always @(posedge clk or posedge reset)
62 begin
63     if (reset == 1)
64     begin
65         output_z <= 32'h0000_0000;
66         abc = 0;
67         xyz = 0;
68         stall_mc1 = 1'b1;
69         stall_mc2 = 1'b0;
70         stall_mc3 = 1'b0;
71         stall_mc4 = 1'b0;
72         stall_mc5 = 1'b0;
73         stall_mc6 = 1'b0;
74         stall_mc7 = 1'b0;
75         stall_mc8 = 1'b0;
76         stall_mc9 = 1'b0;
77         stall_mc10 = 1'b0;
78         a_m_1 <= 0;
79         b_m_1 <= 0;
80         a_e_1 <= 0;
81         b_e_1 <= 0;
82         a_s_1 <= 0;
83         b_s_1 <= 0;
84         a_m_1[26] <= 0;
85     end
86
87     else
88     begin
89
90         if(stall_mc9 == 1'b1)
91         begin
92             output_z <= z_8;
93         end
94         else
95         begin
96             stall_mc10 = 1'b1;
97         end
98     end
99 end
```

```

98
99     if (stall_mc8 == 1'b1)
100     begin
101         z_8[22 : 0] <= z_m_7[22:0];
102         z_8[30 : 23] <= z_e_7[7:0] + 127;
103         z_8[31] <= z_s_7;
104         if ($signed(z_e_7) == -126 && z_m_7[23] == 0)
105         begin
106             z_8[30 : 23] <= 0;
107         end
108         //if overflow occurs, return inf
109         if ($signed(z_e_7) > 127)
110         begin
111             z_8[22 : 0] <= 0;
112             z_8[30 : 23] <= 255;
113             z_8[31] <= z_s_7;
114         end
115         stall_mc9 = 1'b1;
116     end
117
118     if (stall_mc7 == 1'b1)
119     begin
120         if (guard_6 && (round_bit_6 | sticky_6 | z_m_6[0]))
121         begin
122             z_m_7 <= z_m_6 + 1;
123             if (z_m_6 == 24'hfffffff)
124             begin
125                 z_e_7 <= z_e_6 + 1;
126             end
127             else
128             begin
129                 z_e_7 <= z_e_6;
130             end
131         end
132         else
133         begin
134             z_m_7 <= z_m_6;
135             z_s_7 <= z_s_6;
136             z_e_7 <= z_e_6;
137         end
138         z_s_7 <= z_s_6;

```



```
139         stall_mc8 = 1'b1;
140     end
141
142     if (stall_mc6 == 1'b1)
143     begin
144         if ($signed(z_e_5) < -126)
145         begin
146             xyz = -126 - $signed(z_e_5);
147             case(xyz)
148             1:
149             begin
150                 z_e_6 <= z_e_5 + 1;
151                 z_m_6 <= z_m_5 >> 1;
152                 guard_6 <= z_m_5[0];
153                 round_bit_6 <= guard_5;
154                 sticky_6 <= sticky_5 | round_bit_5;
155             end
156             2:
157             begin
158                 z_e_6 <= z_e_5 + 2;
159                 z_m_6 <= z_m_5 >> 2;
160                 guard_6 <= z_m_5[1];
161                 round_bit_6 <= z_m_5[0];
162                 sticky_6 <= sticky_5 | z_m_5[1];
163             end
164             3:
165             begin
166                 z_e_6 <= z_e_5 + 3;
167                 z_m_6 <= z_m_5 >> 3;
168                 guard_6 <= z_m_5[2];
169                 round_bit_6 <= z_m_5[1];
170                 sticky_6 <= sticky_5 | z_m_5[2];
171             end
172             4:
173             begin
174                 z_e_6 <= z_e_5 + 4;
175                 z_m_6 <= z_m_5 >> 4;
176                 guard_6 <= z_m_5[3];
177                 round_bit_6 <= z_m_5[2];
178                 sticky_6 <= sticky_5 | z_m_5[3];
179             end
180         end
181     end
182 end
```

```
180      5:
181      begin
182          z_e_6 <= z_e_5 + 5;
183          z_m_6 <= z_m_5 >> 5;
184          guard_6 <= z_m_5[4];
185          round_bit_6 <= z_m_5[3];
186          sticky_6 <= sticky_5 | z_m_5[4];
187      end
188      6:
189      begin
190          z_e_6 <= z_e_5 + 6;
191          z_m_6 <= z_m_5 >> 6;
192          guard_6 <= z_m_5[5];
193          round_bit_6 <= z_m_5[4];
194          sticky_6 <= sticky_5 | z_m_5[5];
195      end
196      7:
197      begin
198          z_e_6 <= z_e_5 + 7;
199          z_m_6 <= z_m_5 >> 7;
200          guard_6 <= z_m_5[6];
201          round_bit_6 <= z_m_5[5];
202          sticky_6 <= sticky_5 | z_m_5[6];
203      end
204      8:
205      begin
206          z_e_6 <= z_e_5 + 8;
207          z_m_6 <= z_m_5 >> 8;
208          guard_6 <= z_m_5[7];
209          round_bit_6 <= z_m_5[6];
210          sticky_6 <= sticky_5 | z_m_5[7];
211      end
212      9:
213      begin
214          z_e_6 <= z_e_5 + 9;
215          z_m_6 <= z_m_5 >> 9;
216          guard_6 <= z_m_5[8];
217          round_bit_6 <= z_m_5[7];
218          sticky_6 <= sticky_5 | z_m_5[8];
219      end
220      10:
```

```
221         begin
222             z_e_6 <= z_e_5 + 10;
223             z_m_6 <= z_m_5 >> 10;
224             guard_6 <= z_m_5[9];
225             round_bit_6 <= z_m_5[8];
226             sticky_6 <= sticky_5 | z_m_5[9];
227         end
228     11:
229     begin
230         z_e_6 <= z_e_5 + 11;
231         z_m_6 <= z_m_5 >> 11;
232         guard_6 <= z_m_5[10];
233         round_bit_6 <= z_m_5[9];
234         sticky_6 <= sticky_5 | z_m_5[10];
235     end
236     12:
237     begin
238         z_e_6 <= z_e_5 + 12;
239         z_m_6 <= z_m_5 >> 12;
240         guard_6 <= z_m_5[11];
241         round_bit_6 <= z_m_5[10];
242         sticky_6 <= sticky_5 | z_m_5[11];
243     end
244     13:
245     begin
246         z_e_6 <= z_e_5 + 13;
247         z_m_6 <= z_m_5 >> 13;
248         guard_6 <= z_m_5[12];
249         round_bit_6 <= z_m_5[11];
250         sticky_6 <= sticky_5 | z_m_5[12];
251     end
252     14:
253     begin
254         z_e_6 <= z_e_5 + 14;
255         z_m_6 <= z_m_5 >> 14;
256         guard_6 <= z_m_5[13];
257         round_bit_6 <= z_m_5[12];
258         sticky_6 <= sticky_5 | z_m_5[13];
259     end
260     15:
261     begin
```

```
262         z_e_6 <= z_e_5 + 15;
263         z_m_6 <= z_m_5 >> 15;
264         guard_6 <= z_m_5[14];
265         round_bit_6 <= z_m_5[13];
266         sticky_6 <= sticky_5 | z_m_5[14];
267     end
268 16:
269     begin
270         z_e_6 <= z_e_5 + 16;
271         z_m_6 <= z_m_5 >> 16;
272         guard_6 <= z_m_5[15];
273         round_bit_6 <= z_m_5[14];
274         sticky_6 <= sticky_5 | z_m_5[15];
275     end
276 17:
277     begin
278         z_e_6 <= z_e_5 + 17;
279         z_m_6 <= z_m_5 >> 17;
280         guard_6 <= z_m_5[16];
281         round_bit_6 <= z_m_5[15];
282         sticky_6 <= sticky_5 | z_m_5[16];
283     end
284 18:
285     begin
286         z_e_6 <= z_e_5 + 18;
287         z_m_6 <= z_m_5 >> 18;
288         guard_6 <= z_m_5[17];
289         round_bit_6 <= z_m_5[16];
290         sticky_6 <= sticky_5 | z_m_5[17];
291     end
292 19:
293     begin
294         z_e_6 <= z_e_5 + 19;
295         z_m_6 <= z_m_5 >> 19;
296         guard_6 <= z_m_5[18];
297         round_bit_6 <= z_m_5[17];
298         sticky_6 <= sticky_5 | z_m_5[18];
299     end
300 20:
301     begin
302         z_e_6 <= z_e_5 + 20;
```

```

303         z_m_6 <= z_m_5 >> 20;
304         guard_6 <= z_m_5[19];
305         round_bit_6 <= z_m_5[18];
306         sticky_6 <= sticky_5 | z_m_5[19];
307     end
308 21:
309     begin
310         z_e_6 <= z_e_5 + 21;
311         z_m_6 <= z_m_5 >> 21;
312         guard_6 <= z_m_5[20];
313         round_bit_6 <= z_m_5[19];
314         sticky_6 <= sticky_5 | z_m_5[20];
315     end
316 22:
317     begin
318         z_e_6 <= z_e_5 + 22;
319         z_m_6 <= z_m_5 >> 22;
320         guard_6 <= z_m_5[21];
321         round_bit_6 <= z_m_5[20];
322         sticky_6 <= sticky_5 | z_m_5[21];
323     end
324 23:
325     begin
326         z_e_6 <= z_e_5 + 23;
327         z_m_6 <= z_m_5 >> 23;
328         guard_6 <= z_m_5[22];
329         round_bit_6 <= z_m_5[21];
330         sticky_6 <= sticky_5 | z_m_5[22];
331     end
332 endcase
333 z_s_6 <= z_s_5;
334 stall_mc7 = 1'b1;
335 end
336 else
337     begin
338         z_e_6 <= z_e_5;
339         z_m_6 <= z_m_5;
340         guard_6 <= guard_5;
341         round_bit_6 <= round_bit_5;
342         sticky_6 <= sticky_5;
343         z_s_6 <= z_s_5;

```

```

344         stall_mc7 = 1'b1;
345     end
346 end
347
348 if(stall_mc5 == 1'b1)
349 begin
350     casex(z_m_4)
351     24'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx:
352     begin
353         z_m_5 <= z_m_4;
354         z_e_5 <= z_e_4;
355         z_m_5[0] <= z_m_4[0];
356         guard_5 <= guard_4;
357         round_bit_5 <= round_bit_4;
358         z_s_5 <= z_s_4;
359         sticky_5 <= sticky_4;
360         stall_mc6 = 1'b1;
361     end
362     24'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx:
363     begin
364         z_m_5 <= z_m_4 << 1;
365         z_e_5 <= z_e_4 - 1;
366         z_m_5[0] <= guard_4;
367         guard_5 <= round_bit_4;
368         round_bit_5 <= 0;
369     end
370     24'b001x_xxxx_xxxx_xxxx_xxxx_xxxx:
371     begin
372         z_m_5 <= z_m_4 << 2;
373         z_e_5 <= z_e_4 - 2;
374         z_m_5[0] <= round_bit_4;
375         guard_5 <= 0;
376         round_bit_5 <= 0;
377     end
378     24'b0001_xxxx_xxxx_xxxx_xxxx_xxxx:
379     begin
380         z_m_5 <= z_m_4 << 3;
381         z_e_5 <= z_e_4 - 3;
382         z_m_5[0] <= 0;
383         guard_5 <= 0;
384         round_bit_5 <= 0;

```

```

385     end
386 24'b0000_1xxx_xxxx_xxxx_xxxx_xxxx:
387 begin
388     z_m_5 <= z_m_4 << 4;
389     z_e_5 <= z_e_4 - 4;
390     z_m_5[0] <= 0;
391     guard_5 <= 0;
392     round_bit_5 <= 0;
393 end
394 24'b0000_01xx_xxxx_xxxx_xxxx_xxxx:
395 begin
396     z_m_5 <= z_m_4 << 5;
397     z_e_5 <= z_e_4 - 5;
398     z_m_5[0] <= 0;
399     guard_5 <= 0;
400     round_bit_5 <= 0;
401 end
402 24'b0000_001x_xxxx_xxxx_xxxx_xxxx:
403 begin
404     z_m_5 <= z_m_4 << 6;
405     z_e_5 <= z_e_4 - 6;
406     z_m_5[0] <= 0;
407     guard_5 <= 0;
408     round_bit_5 <= 0;
409 end
410 24'b0000_0001_xxxx_xxxx_xxxx_xxxx:
411 begin
412     z_m_5 <= z_m_4 << 7;
413     z_e_5 <= z_e_4 - 7;
414     z_m_5[0] <= 0;
415     guard_5 <= 0;
416     round_bit_5 <= 0;
417 end
418 24'b0000_0000_1xxx_xxxx_xxxx_xxxx:
419 begin
420     z_m_5 <= z_m_4 << 8;
421     z_e_5 <= z_e_4 - 8;
422     z_m_5[0] <= 0;
423     guard_5 <= 0;
424     round_bit_5 <= 0;
425 end

```

```

426      24'b0000_0000_01xx_xxxx_xxxx_xxxx:
427      begin
428          z_m_5 <= z_m_4 << 9;
429          z_e_5 <= z_e_4 - 9;
430          z_m_5[0] <= 0;
431          guard_5 <= 0;
432          round_bit_5 <= 0;
433      end
434      24'b0000_0000_001x_xxxx_xxxx_xxxx:
435      begin
436          z_m_5 <= z_m_4 << 10;
437          z_e_5 <= z_e_4 - 10;
438          z_m_5[0] <= 0;
439          guard_5 <= 0;
440          round_bit_5 <= 0;
441      end
442      24'b0000_0000_0001_xxxx_xxxx_xxxx:
443      begin
444          z_m_5 <= z_m_4 << 11;
445          z_e_5 <= z_e_4 - 11;
446          z_m_5[0] <= 0;
447          guard_5 <= 0;
448          round_bit_5 <= 0;
449      end
450      24'b0000_0000_0000_1xxx_xxxx_xxxx:
451      begin
452          z_m_5 <= z_m_4 << 12;
453          z_e_5 <= z_e_4 - 12;
454          z_m_5[0] <= 0;
455          guard_5 <= 0;
456          round_bit_5 <= 0;
457      end
458      24'b0000_0000_0000_01xx_xxxx_xxxx:
459      begin
460          z_m_5 <= z_m_4 << 13;
461          z_e_5 <= z_e_4 - 13;
462          z_m_5[0] <= 0;
463          guard_5 <= 0;
464          round_bit_5 <= 0;
465      end
466      24'b0000_0000_0000_001x_xxxx_xxxx:

```



```

467         begin
468             z_m_5 <= z_m_4 << 14;
469             z_e_5 <= z_e_4 - 14;
470             z_m_5[0] <= 0;
471             guard_5 <= 0;
472             round_bit_5 <= 0;
473         end
474 24'b0000_0000_0000_0001_xxxx_xxxx:
475     begin
476         z_m_5 <= z_m_4 << 15;
477         z_e_5 <= z_e_4 - 15;
478         z_m_5[0] <= 0;
479         guard_5 <= 0;
480         round_bit_5 <= 0;
481     end
482 24'b0000_0000_0000_0000_1xxx_xxxx:
483     begin
484         z_m_5 <= z_m_4 << 16;
485         z_e_5 <= z_e_4 - 16;
486         z_m_5[0] <= 0;
487         guard_5 <= 0;
488         round_bit_5 <= 0;
489     end
490 24'b0000_0000_0000_0000_01xx_xxxx:
491     begin
492         z_m_5 <= z_m_4 << 17;
493         z_e_5 <= z_e_4 - 17;
494         z_m_5[0] <= 0;
495         guard_5 <= 0;
496         round_bit_5 <= 0;
497     end
498 24'b0000_0000_0000_0000_001x_xxxx:
499     begin
500         z_m_5 <= z_m_4 << 18;
501         z_e_5 <= z_e_4 - 18;
502         z_m_5[0] <= 0;
503         guard_5 <= 0;
504         round_bit_5 <= 0;
505     end
506 24'b0000_0000_0000_0000_0001_xxxx:
507     begin

```

```

508         z_m_5 <= z_m_4 << 19;
509         z_e_5 <= z_e_4 - 19;
510         z_m_5[0] <= 0;
511         guard_5 <= 0;
512         round_bit_5 <= 0;
513     end
514 24'b0000_0000_0000_0000_0000_1xxx:
515 begin
516     z_m_5 <= z_m_4 << 20;
517     z_e_5 <= z_e_4 - 20;
518     z_m_5[0] <= 0;
519     guard_5 <= 0;
520     round_bit_5 <= 0;
521 end
522 24'b0000_0000_0000_0000_0000_01xx:
523 begin
524     z_m_5 <= z_m_4 << 21;
525     z_e_5 <= z_e_4 - 21;
526     z_m_5[0] <= 0;
527     guard_5 <= 0;
528     round_bit_5 <= 0;
529 end
530 24'b0000_0000_0000_0000_0000_001x:
531 begin
532     z_m_5 <= z_m_4 << 22;
533     z_e_5 <= z_e_4 - 22;
534     z_m_5[0] <= 0;
535     guard_5 <= 0;
536     round_bit_5 <= 0;
537 end
538 24'b0000_0000_0000_0000_0000_0001:
539 begin
540     z_m_5 <= z_m_4 << 23;
541     z_e_5 <= z_e_4 - 23;
542     z_m_5[0] <= 0;
543     guard_5 <= 0;
544     round_bit_5 <= 0;
545 end
546 endcase
547 z_s_5 <= z_s_4;
548 sticky_5 <= sticky_4;

```

```
549         stall_mc6 = 1'b1;
550     end
551
552     if(stall_mc4 == 1'b1)
553     begin
554         if (sum_3[27])
555         begin
556             z_m_4 <= sum_3[27:4];
557             guard_4 <= sum_3[3];
558             round_bit_4 <= sum_3[2];
559             sticky_4 <= sum_3[1] | sum_3[0];
560             z_e_4 <= z_e_3 + 1;
561         end
562         else
563         begin
564             z_m_4 <= sum_3[26:3];
565             z_e_4 <= z_e_3;
566             guard_4 <= sum_3[2];
567             round_bit_4 <= sum_3[1];
568             sticky_4 <= sum_3[0];
569         end
570         z_s_4 <= z_s_3;
571         stall_mc5 = 1'b1;
572     end
573
574     if(stall_mc3 == 1'b1)
575     begin
576         z_e_3 <= a_e_2;
577         if (a_s_2 == b_s_2)
578         begin
579             sum_3 <= a_m_2 - b_m_2;
580             z_s_3 <= a_s_2;
581         end
582         else
583         begin
584             if (a_m_2 >= b_m_2)
585             begin
586                 sum_3 <= a_m_2 - b_m_2;
587                 z_s_3 <= a_s_2;
588             end
589             else
```

```

590         begin
591             sum_3 <= b_m_2 - a_m_2;
592             z_s_3 <= b_s_2;
593         end
594     end
595
596     stall_mc4 = 1'b1;
597 end
598
599 if (stall_mc2 == 1'b1)
600 begin
601     if ($signed(a_e_1) > $signed(b_e_1))
602     begin
603         abc = a_e_1 - b_e_1;
604         case (abc)
605             8'd1: b_m_2 <= b_m_1 >> 1;
606             8'd2: b_m_2 <= b_m_1 >> 2;
607             8'd3: b_m_2 <= b_m_1 >> 3;
608             8'd4: b_m_2 <= b_m_1 >> 4;
609             8'd5: b_m_2 <= b_m_1 >> 5;
610             8'd6: b_m_2 <= b_m_1 >> 6;
611             8'd7: b_m_2 <= b_m_1 >> 7;
612             8'd8: b_m_2 <= b_m_1 >> 8;
613             8'd9: b_m_2 <= b_m_1 >> 9;
614             8'd10: b_m_2 <= b_m_1 >> 10;
615             8'd11: b_m_2 <= b_m_1 >> 11;
616             8'd12: b_m_2 <= b_m_1 >> 12;
617             8'd13: b_m_2 <= b_m_1 >> 13;
618             8'd14: b_m_2 <= b_m_1 >> 14;
619             8'd15: b_m_2 <= b_m_1 >> 15;
620             8'd16: b_m_2 <= b_m_1 >> 16;
621             8'd17: b_m_2 <= b_m_1 >> 17;
622             8'd18: b_m_2 <= b_m_1 >> 18;
623             8'd19: b_m_2 <= b_m_1 >> 19;
624             8'd20: b_m_2 <= b_m_1 >> 20;
625             8'd21: b_m_2 <= b_m_1 >> 21;
626             8'd22: b_m_2 <= b_m_1 >> 22;
627             8'd23: b_m_2 <= b_m_1 >> 23;
628             8'd24: b_m_2 <= b_m_1 >> 24;
629             8'd25: b_m_2 <= b_m_1 >> 25;
630             8'd26: b_m_2 <= b_m_1 >> 26;

```

```
631         default :
632         begin
633             b_m_2 <= 1;
634             b_e_2 <= a_e_1;
635         end
636     endcase
637     a_m_2 <= a_m_1;
638     b_e_2 <= a_e_1;
639     a_e_2 <= a_e_1;
640     stall_mc3 = 1'b1;
641 end
642 else if ($signed(a_e_1) < $signed(b_e_1))
643 begin
644     abc = b_e_1 - a_e_1;
645     case(abc)
646     8'd1: a_m_2 <= a_m_1 >> 1;
647     8'd2: a_m_2 <= a_m_1 >> 2;
648     8'd3: a_m_2 <= a_m_1 >> 3;
649     8'd4: a_m_2 <= a_m_1 >> 4;
650     8'd5: a_m_2 <= a_m_1 >> 5;
651     8'd6: a_m_2 <= a_m_1 >> 6;
652     8'd7: a_m_2 <= a_m_1 >> 7;
653     8'd8: a_m_2 <= a_m_1 >> 8;
654     8'd9: a_m_2 <= a_m_1 >> 9;
655     8'd10: a_m_2 <= a_m_1 >> 10;
656     8'd11: a_m_2 <= a_m_1 >> 11;
657     8'd12: a_m_2 <= a_m_1 >> 12;
658     8'd13: a_m_2 <= a_m_1 >> 13;
659     8'd14: a_m_2 <= a_m_1 >> 14;
660     8'd15: a_m_2 <= a_m_1 >> 15;
661     8'd16: a_m_2 <= a_m_1 >> 16;
662     8'd17: a_m_2 <= a_m_1 >> 17;
663     8'd18: a_m_2 <= a_m_1 >> 18;
664     8'd19: a_m_2 <= a_m_1 >> 19;
665     8'd20: a_m_2 <= a_m_1 >> 20;
666     8'd21: a_m_2 <= a_m_1 >> 21;
667     8'd22: a_m_2 <= a_m_1 >> 22;
668     8'd23: a_m_2 <= a_m_1 >> 23;
669     8'd24: a_m_2 <= a_m_1 >> 24;
670     8'd25: a_m_2 <= a_m_1 >> 25;
671     8'd26: a_m_2 <= a_m_1 >> 26;
```

```

672         default :
673         begin
674             a_m_2 <= 1;
675             a_e_2 <= b_e_1;
676         end
677     endcase
678     b_m_2 <= b_m_1;
679     a_e_2 <= b_e_1;
680     b_e_2 <= b_e_1;
681     stall_mc3 = 1'b1;
682 end
683 else
684 begin
685     a_m_2 <= a_m_1;
686     b_m_2 <= b_m_1;
687     a_e_2 <= a_e_1;
688     b_e_2 <= b_e_1;
689     stall_mc3 = 1'b1;
690 end
691 a_s_2 <= a_s_1;
692 b_s_2 <= b_s_1;
693 end
694
695 if(stall_mc1 == 1'b1)
696 begin
697     a_m_1 <= {input_a[22 : 0], 3'd0};
698     b_m_1 <= {input_b[22 : 0], 3'd0};
699     a_e_1 <= input_a[30 : 23] - 127;
700     b_e_1 <= input_b[30 : 23] - 127;
701     a_s_1 <= input_a[31];
702     b_s_1 <= input_b[31];
703     a_m_1[26] <= 1;
704
705     if(input_b[30:23] == 0)
706     begin
707         b_e_1 <= -126;
708     end
709     else
710     begin
711         b_m_1[26] <= 1;
712     end

```

```
713         stall_mc2 = 1'b1;
714     end
715 end
716 end
717 endmodule
```

I.5 Subtractor Module

```
1 module mvk_subtractor (
2     reset ,
3     clk ,
4     scan_in0 ,
5     scan_en ,
6     test_mode ,
7     number_1 ,
8     number_2 ,
9     answer ,
10    scan_out0
11 );
12
13 input
14     reset ,           // system reset
15     clk ;             // system clock
16
17 input
18     scan_in0 ,        // test scan mode data input
19     scan_en ,         // test scan mode enable
20     test_mode ;       // test mode select
21
22 output
23     scan_out0 ;       // test scan mode data output
24
25 input
26     [11:0] number_1 ;
27
28 input
29     [10:0] number_2 ;
30
```

```
31 output
32     [31:0] answer;
33
34 wire
35     [23:0] inverse_2;
36
37 wire
38     [23:0] sum;
39
40 wire
41     [23:0] carry;
42
43 wire
44     [23:0] big_number;
45
46 wire
47     [23:0] number_2_inverse;
48
49 wire
50     [7:0] exponent;
51
52 wire
53     [22:0] a_m;
54
55 //big number contains the yh part.
56 //The input number is padded with 10'b0 at the end
57 assign big_number = {1'b1, number_1[11], number_1[10], number_1
    [9], number_1[8], number_1[7], number_1[6], number_1[5],
    number_1[4], number_1[3], number_1[2], number_1[1], number_1
    [0], 11'b00000000000};
58
59 //inverse_2 contains the inverse of input number 2 which is yl
    and padded with 1's at the start.
60 assign inverse_2 = {2'b00, number_2, 11'b00000000000};
61
62 assign answer = (reset == 1'b1) ? 32'h0000_0000 : {1'b0, exponent
    , a_m};
63
64 assign sum = big_number - inverse_2;
65 assign a_m = (sum <= 24'b0000_0000_0000_0000_0000_0000) ? 24'd0
    :
```



```

66      (sum <= 24'b0000_0000_0000_0000_0000_0001) ? sum <<
        23 :
67      (sum <= 24'b0000_0000_0000_0000_0000_0011) ? sum <<
        22 :
68      (sum <= 24'b0000_0000_0000_0000_0000_0111) ? sum <<
        21 :
69      (sum <= 24'b0000_0000_0000_0000_0000_1111) ? sum <<
        20 :
70      (sum <= 24'b0000_0000_0000_0000_0001_1111) ? sum <<
        19 :
71      (sum <= 24'b0000_0000_0000_0000_0011_1111) ? sum <<
        18 :
72      (sum <= 24'b0000_0000_0000_0000_0111_1111) ? sum <<
        17 :
73      (sum <= 24'b0000_0000_0000_0000_1111_1111) ? sum <<
        16 :
74      (sum <= 24'b0000_0000_0000_0001_1111_1111) ? sum <<
        15 :
75      (sum <= 24'b0000_0000_0000_0011_1111_1111) ? sum <<
        14 :
76      (sum <= 24'b0000_0000_0000_0111_1111_1111) ? sum <<
        13 :
77      (sum <= 24'b0000_0000_0000_1111_1111_1111) ? sum <<
        12 :
78      (sum <= 24'b0000_0000_0001_1111_1111_1111) ? sum <<
        11 :
79      (sum <= 24'b0000_0000_0011_1111_1111_1111) ? sum <<
        10 :
80      (sum <= 24'b0000_0000_0111_1111_1111_1111) ? sum <<
        9 :
81      (sum <= 24'b0000_0000_1111_1111_1111_1111) ? sum <<
        8 :
82      (sum <= 24'b0000_0001_1111_1111_1111_1111) ? sum <<
        7 :
83      (sum <= 24'b0000_0011_1111_1111_1111_1111) ? sum <<
        6 :
84      (sum <= 24'b0000_0111_1111_1111_1111_1111) ? sum <<
        5 :
85      (sum <= 24'b0000_1111_1111_1111_1111_1111) ? sum <<
        4 :

```

```

86      (sum <= 24'b0001_1111_1111_1111_1111) ? sum <<
      3 :
87      (sum <= 24'b0011_1111_1111_1111_1111) ? sum <<
      2 :
88      (sum <= 24'b0111_1111_1111_1111_1111) ? sum <<
      1 :
89      (sum <= 24'b1111_1111_1111_1111_1111) ? sum <<
      0 : 24'd0;
90
91  assign exponent = (sum <= 24'b0000_0000_0000_0000_0000_0000) ?
      8'd0 :
92      (sum <= 24'b0000_0000_0000_0000_0000_0001) ?
      8'd150 :
93      (sum <= 24'b0000_0000_0000_0000_0000_0011) ?
      8'd149 :
94      (sum <= 24'b0000_0000_0000_0000_0000_0111) ?
      8'd148 :
95      (sum <= 24'b0000_0000_0000_0000_0000_1111) ?
      8'd147 :
96      (sum <= 24'b0000_0000_0000_0000_0001_1111) ?
      8'd146 :
97      (sum <= 24'b0000_0000_0000_0000_0011_1111) ?
      8'd145 :
98      (sum <= 24'b0000_0000_0000_0000_0111_1111) ?
      8'd144 :
99      (sum <= 24'b0000_0000_0000_0000_1111_1111) ?
      8'd143 :
100     (sum <= 24'b0000_0000_0000_0001_1111_1111) ?
      8'd142 :
101     (sum <= 24'b0000_0000_0000_0011_1111_1111) ?
      8'd141 :
102     (sum <= 24'b0000_0000_0000_0111_1111_1111) ?
      8'd140 :
103     (sum <= 24'b0000_0000_0000_1111_1111_1111) ?
      8'd139 :
104     (sum <= 24'b0000_0000_0001_1111_1111_1111) ?
      8'd138 :
105     (sum <= 24'b0000_0000_0011_1111_1111_1111) ?
      8'd137 :
106     (sum <= 24'b0000_0000_0111_1111_1111_1111) ?
      8'd136 :

```

```
107      (sum <= 24'b0000_0000_1111_1111_1111_1111) ?  
      8'd135 :  
108      (sum <= 24'b0000_0001_1111_1111_1111_1111) ?  
      8'd134 :  
109      (sum <= 24'b0000_0011_1111_1111_1111_1111) ?  
      8'd133 :  
110      (sum <= 24'b0000_0111_1111_1111_1111_1111) ?  
      8'd132 :  
111      (sum <= 24'b0000_1111_1111_1111_1111_1111) ?  
      8'd131 :  
112      (sum <= 24'b0001_1111_1111_1111_1111_1111) ?  
      8'd130 :  
113      (sum <= 24'b0011_1111_1111_1111_1111_1111) ?  
      8'd129 :  
114      (sum <= 24'b0111_1111_1111_1111_1111_1111) ?  
      8'd128 :  
115      (sum <= 24'b1111_1111_1111_1111_1111_1111) ?  
      8'd127 : 8'd0;  
116  
117 endmodule // mvk_subtractor
```
